A Manual for the Cyber Security Modeling Language

Hannes Holm, Mathias Ekstedt, Teodor Sommestad, Matus Korman
Department of Industrial Information and Control Systems, Royal Institute of Technology, 100 44 Stockholm, Sweden

Abstract
The Cyber Security Modeling Language (CySeMoL) is an attack graph tool that can be used to estimate the cyber security of enterprise architectures. CySeMoL includes theory on how attacks and defenses relate quantitatively; thus, users must only model their assets and how these are connected in order to enable calculations. This report functions as a manual to facilitate practical usage and understanding of CySeMoL.

Date: 2014-05-27

Changes from CySeMoL v2.0 to v2.1

(1) Softened coloring scheme of calculation results

Motivation. This change is introduced due to comfort.

Effect. The change impacts how calculation results are visualized in an object model.

(2) Added possibility to input injective evidence in object models, for faster calculation using rejection sampling and Metropolis-Hastings sampling

Motivation. Previously, providing much evidence in models, lead to slow calculations when using rejection and Metropolis-Hastings sampling. To address the performance problem, a new concept of injective evidence was introduced to CySeMoL.

Function. Unlike the original (classical) concept of evidence, injective evidence unconditionally overrides the derivation of an attribute's value, for which the evidence is provided. Hence, and unlike for the classical evidence, a whole sample across the model is not rejected if a derived attribute value lacks consistency with the provided injective evidence.

Usage. Similarly to inputting the classical evidence, the user needs to select a defense mechanism (or an attack step) of an asset in a model. Subsequently, the property browser, usually to the right from the main modeling control, will include properties named *Functioning_EvidenceToInject* and *Functioning_InjectEvidence* (for attack steps, they would be named *Likelihood_EvidenceToInject* and *Likelihood_InjectEvidence*, respectively). The latter property, *InjectEvidence*, indicates whether injective evidence should be used. The former property, *EvidenceToInject*, indicates what specific evidence should be injected. Both properties contain sub-property called *Evidence*, which contains an OCL expression that has to yield true, false or no value (=> false)). For example, in order to set positive injective evidence for that a specific installation of an operating system is fully patched, the user needs to select that defense mechanism (i.e., *HasAllPatches* on the operating system), and set its properties *Functioning_EvidenceToInject*.*Evidence* to true, and *Functioning_InjectEvidence*.*Evidence* to true. Similarly, in order to set negative injective evidence for that a security awareness program is conducted for some people in the architecture, the user needs to select that security awareness program's defense mechanism called *TrainingConducted*, and set its properties *Functioning_EvidenceToInject*.*Evidence* to false, and *Functioning_InjectEvidence*.*Evidence* to true.

Note. Although injective evidence is not valid to use in all cases, it is safe to use in cases of direct derivation of value. For example, one can use injective evidence instead of the classical one for the availability of defense mechanisms that one ultimately knows are present (or absent), or where one can specify an ultimately trusted probability distribution of their availability. Injective evidence should not be used on any attributes that are being calculated based on the value of another attribute in the object model, because doing so could make the calculation result erroneous. It is generally invalid (unsafe) to use injective evidence for attack steps.

(3) Correction of default availability of defense mechanisms (50%-50% now) – for defense mechanisms that do not override this, and those for which no injective evidence is provided

Motivation. In previous version of CySeMoL, the default generic availability of defense mechanisms was set to true, which disallowed simulating full uncertainty of the availability of defense mechanisms that do not further override the value by their own derivation, which affects a subset of defense mechanisms in CySeMoL. Full uncertainty presupposes that there is an equal (50%-50%) chance of the availability being true as false, which the default generic derivation.

Effect. Improvement in calculation.

Changes from CySemoL v2.1 to v2.2

(1) Correction (bugfix) of evaluation of attack step *GuessCredentialsOnline* in asset *PasswordAccount*

Motivation. An incorrect result was observed when trying different configurations of defenses of a *PasswordAuthenticationMechanism* (PAM) connected to an *AccessControlPoint* (ACP), which was further connected to a *PasswordAccount* (PA). The configuration {*BackoffTechnique*=true; *DefaultPasswordsRemoved*=true; *Functioning*=true; *HashedRepository*=true; *HashedRepositorySalted*=false; *ProactivePasswordChecker*=false} at PAM led to the probability of reaching PA's *GuessCredentialsOnline* equal to 0.6, but setting PAM's *DefaultPasswordChecker* to true (from false, which is an obvious security improvement), PA's *GuessCredentialsOnline* could be reached with the probability of 1 (which is an absurd result). Instead, the probability should have been 0, according to the manual and underlying theory.

Effect. Correction of calculation of the attack step *GuessCredentialsOnline* at the asset *PasswordAccount*.

A Manual for the Cyber Security Modeling Language

Hannes Holm, Mathias Ekstedt, Teodor Sommestad, Matus Korman

Department of Industrial Information and Control Systems, Royal Institute of Technology, 100 44 Stockholm, Sweden

Abstract

The Cyber Security Modeling Language (CySeMoL) is an attack graph tool that can be used to estimate the cyber security of enterprise architectures. CySeMoL includes theory on how attacks and defenses relate quantitatively; thus, users must only model their assets and how these are connected in order to enable calculations. This report functions as a manual to facilitate practical usage and understanding of CySeMoL.

Keywords: Cyber security, security metrics, attack graphs, SCADA systems

1. Introduction

Information Technology (IT) is today a cornerstone of next to all business as IT applications handle everything from management of critical data to control of physical processes such as the power grid. Considerable effort is thus spent by both researchers and practitioners to preserve IT systems in a reliable and predictable state. This is however a difficult topic to manage as a modern IT architecture typically is composed of a large number of systems, processes and individuals connected to form a complex system-of-systems (hereafter refered to simply as *system*). Threats towards the state of the system arise from errors made both during the development and the maintenance of employed IT.

The presence of individuals determined to exploit these errors to conduct unauthorized activity in the system adds another layer to the complexity of the problem. To estimate the vulnerability of a system, an enormous amount of factors need be considered. It is not sufficient to address all vulnerabilities within it - there is also a need to understand how these vulnerabilities relate.

Email address: mathiase@ics.kth.se (Mathias Ekstedt)

Date of revision *May 28, 2014*

Consequently, it is a difficult task for enterprise decision makers to effectively manage the cyber security of their system. A common means of estimating the cyber security of their system in practice is to consult experts, e.g., network penetration testers. While consulting experts certainly is valuable, resulting estimates come with three significant delimitations: they are only valid for 1) the time that they were carried out, 2) the parts of the enterprise architecture that were studied by the expert, and 3) the competence of the consulted expert. These delimitations are especially problematic given the dynamic nature of enterprise IT systems and the lack of resources available for analyses.

Enterprise decision makers are thus in need of tools that can help estimate the cyber security of their system in an easy-to-understand fashion. While there are various tools available for this purpose, most suffer from being either too vague, and thus ultimately subjective [1] (e.g., Common Criteria [2], OCTAVE [3], CORAS [4] and the model by Breu et al. [5]), or too limited in terms of scope (e.g., MulVAL [6, 7], NetSPA [8] or TVA-tool [9]). With the shortcomings of existing tools in mind, researchers at the department of Industrial information and Control Systems (ICS) at the Royal Institute of Technology (KTH) in Stockholm, Sweden, developed a new tool denoted the Cyber Security Modeling Language (CySeMoL) [10].

CySeMoL is a modeling framework and calculation engine for estimating the cyber security of enterprise-level system architectures [10]. CySeMoL includes theory on how attacks and defenses relate quantitatively; thus, security expertise is not required from its users. Users must only model their system architecture (e.g., services, operating systems, networks, and personnel) and specify their characteristics (e.g., if an operating system has a host firewall enabled) in order to enable calculations.

The purpose of this report is to describe the content of CySeMoL. In other words, it functions as a manual to facilitate practical usage and understanding of CySeMoL.

The remainder of this report is structured as follows: Section 2 describes the Predictive, Probabilistic Architecture Modeling Framework (P^2AMF), the framework used to construct attack graphs and calculate vulnerability estimates. Section 3 describes the software tool that CySeMoL has been implemented in and how it is used. Section 4 describes the overall logical of CySeMoL. Section 5 - Section 28 constitute the core of the report and describe the many concepts of CySeMoL in detail. Finally, Section 29 provides information about screencasts of CySeMoL.

2. P^{2}AMF: Predictive, Probabilistic Architecture Modeling Framework

To enable modeling and calculation of the vulnerability of a system-of-systems, there is need of a framework that dictates how attacks and defenses relate. For this purpose, CySeMoL employs the Predictive, Probabilistic Architecture Modeling Framework (P^{2}AMF) [11] - this framework is briefly described in the remainder of the present section. The CySeMoL implementation of P^{2}AMF is described in Section 4.1.

P^{2}AMF is an extension of the Object Constraint Language (OCL) [12] for probabilistic assessment and prediction of system properties. The main feature of P^{2}AMF is its ability to express uncertainties of objects, relations and attributes in Unified Modeling Language (UML) models and perform probabilistic assessments incorporating these uncertainties. A typical usage of P^{2}AMF would be to create a model for predicting some phenomenon, e.g., the availability of a certain node.

In P^{2}AMF, two kinds of uncertainty are introduced. First, attributes may be stochastic. When attributes are instantiated, their values are expressed as probability distributions. Second, the existence of objects and relationships may be uncertain. It may be the case that one no longer knows whether a specific node is still in service, or not. This is a case of object existence uncertainty. Such uncertainty is specified using an existence attribute `E` that is mandatory for all classes (here *class* refers to the typical object-oriented meaning of the word). For instance, the probability P that the node instance `Database server` exists might be $P = P(\texttt{Database server.E}) = 0.8$ (see Figure 1).

In other words, there is an 80% likelihood that `Database server` still exists. It might also be uncertain whether `Database server` is still assigned to a specific infrastructure function, i.e., whether there is a connection between the server and the infrastructure function. Similarly, relationship uncertainty is specified with an existence attribute `E` on the relationships. The probabilistic aspects are considered in a Monte-Carlo fashion: First, the user specifies a desired number of samples. Thereafter, a set of object models corresponding to the chosen sample size is created. The stochastic variables of the class model are instantiated with instance values according to their respective designated distribution. This includes the existence of classes and relationships, which are instantiated on a frequency depicted by the corresponding probability distributions. Then, each P^{2}AMF statement is transformed into a proper OCL statement and can be evaluated by the OCL parser. Once the evaluation of all samples has been performed, results are aggregated and visualized according to the design of the class model.

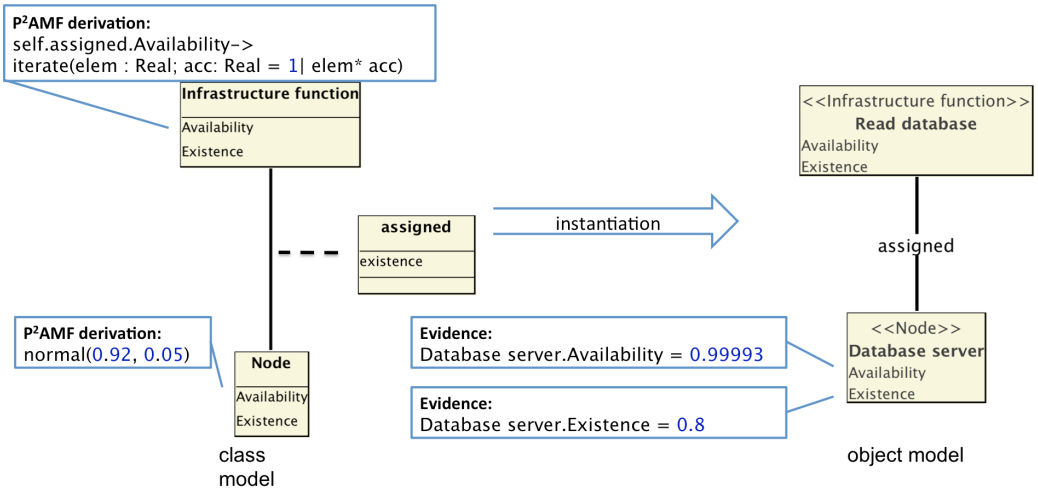

Figure 1: Example P^{2}AMF class model and object model.

3. The Enterprise Architecture Analysis Tool

CySeMoL has been implemented in a software tool, the Enterprise Architecture Analysis Tool (EAAT) that enables a user-friendly interace for both modeling and analysis. General information on how to use EAAT can be found at *www.ics.kth.se/eaat/manuals*; this section describes the specifics of EAAT that concern CySeMoL.

3.1. Creation of an object model

After opening up the CySeMoL class model in the EAAT Object Modeler (OM), the user is greeted by the concepts and templates given in Figure 2. In CySeMoL, a *concept* refers to an attack step, defense or asset. While these can be modeled manually by a user, it is not recommended to do so. Rather, a user of CySeMoL should depict *templates* that relate attack steps and defenses to assets in a meaningful and automatic fashion (cf. Section 4.1). An example template is `OperatingSystem`, which consists of one asset, seven defenses and nine attack steps (cf. Section 6); the connections between these are already pre-defined in the `OperatingSystem` template. To model a template, there is simply a need to drag-and-drop it from the list of templates to the view canvas in the center of the screen.

Different templates can be connected in varied means to depict different types of relations. For instance, an application server can either be a terminal that allows users to interface core functionality of an operating system (e.g., a Secure Shell

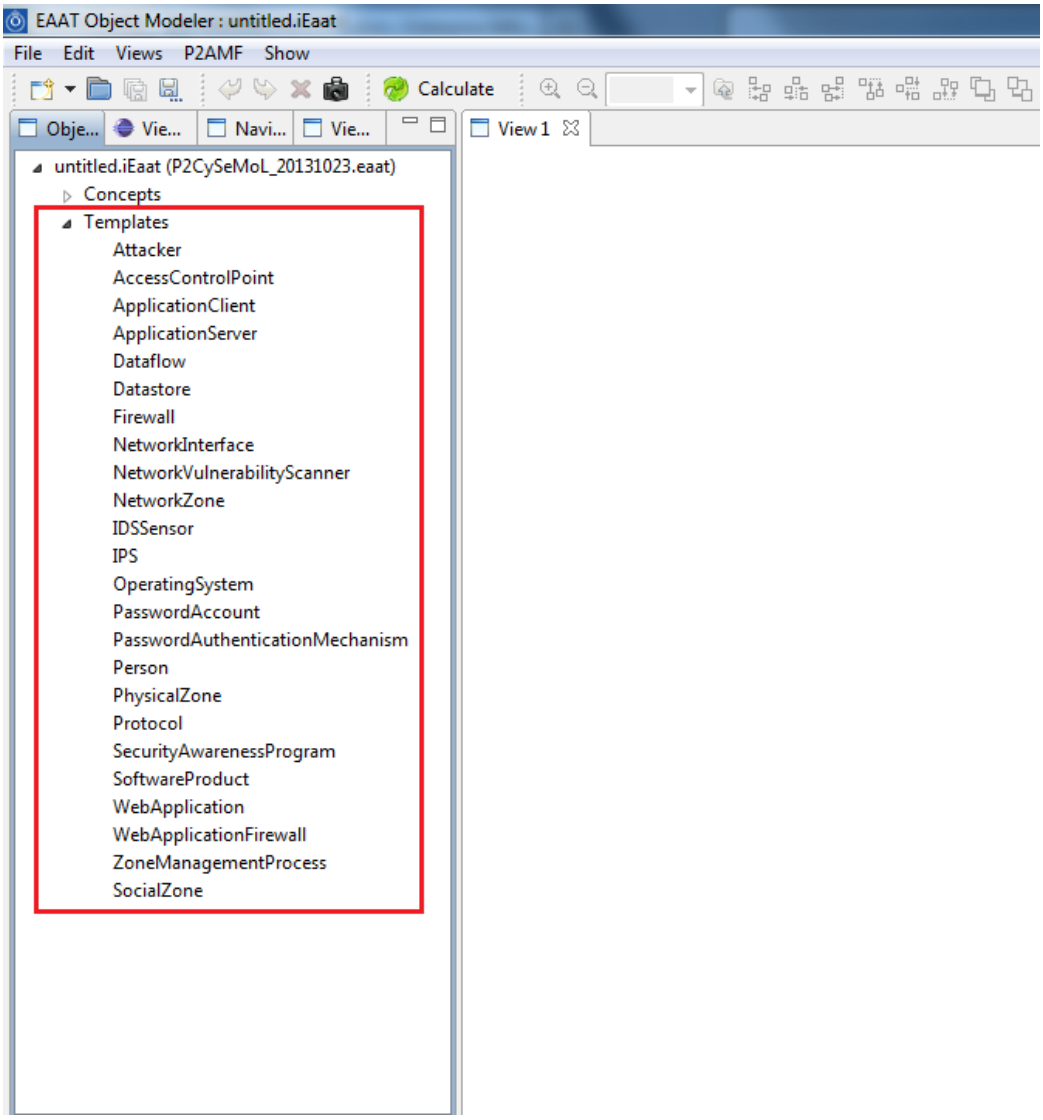

Figure 2: EAAT templates for CySeMoL.

(SSH)) or not (e.g., a Hypertext Transfer Protocol (HTTP) server). When a user connects two templates (by holding ALT and left-clicking), a dialogue such as shown in Figure 3 will appear. This dialogue denotes what connections that are possible for the two templates. The user should then choose the connection type that conforms to his or her context. If no connections are possible, no dialogue will be shown. Similarly, if only a single connection is possible, this will be

depicted automatically without any dialogue.

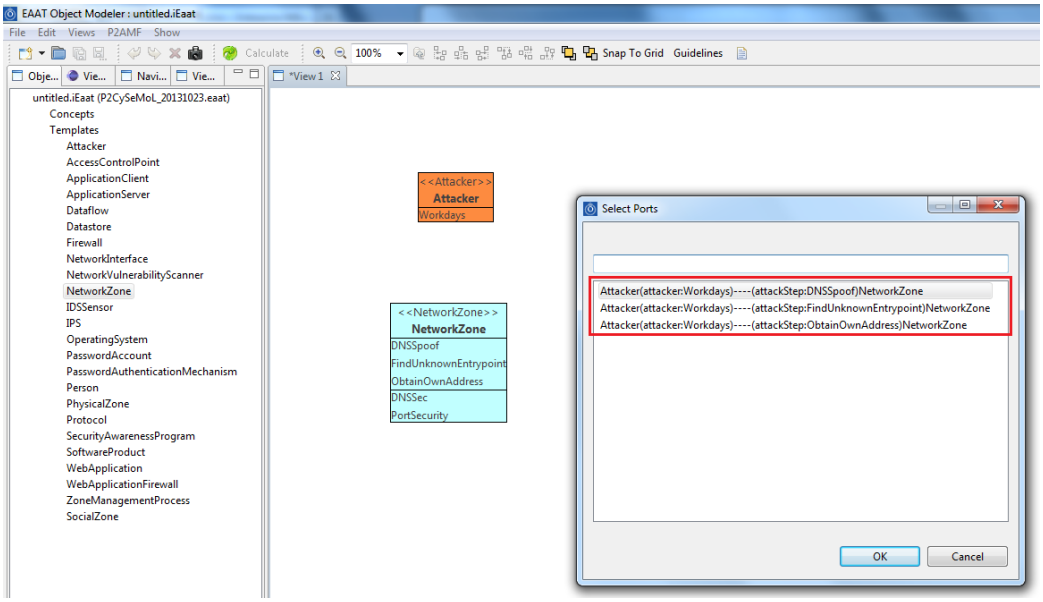

Figure 3: EAAT connection dialogue.

In EAAT, the same object can be visualized in different *views*. Here, a view correspond to a certain collection of objects and relations that should be visible to the user. A view can be customized to show only that which is relevant to a user through the view properties tab (cf. Figure 4)

As shown in Figure 5-6, the states of defenses and attack steps can be manually inputed by a user. This is however not recommended for attack steps as the states of these should be calculated by CySeMoL. Furthermore, it is recommended, but not required, for defenses as they either have default-values defined in CySeMoL or are dependent on the availability of other defenses in an object model (cf. Section 4).

3.2. Executing calculations

To conduct calculations on vulnerability severity, there is first a need to define the calculation parameters of an object model. To reach the calculation configuration dialogue, the user should first left-click “Configurations” within the “P2AMF” tab (cf. Figure 7).

From within this dialogue, the user should select the sampling method denoted “FORWARD_EVIDENCE_INJECTION_SAMPLING” (cf. Figure 8). Then, the

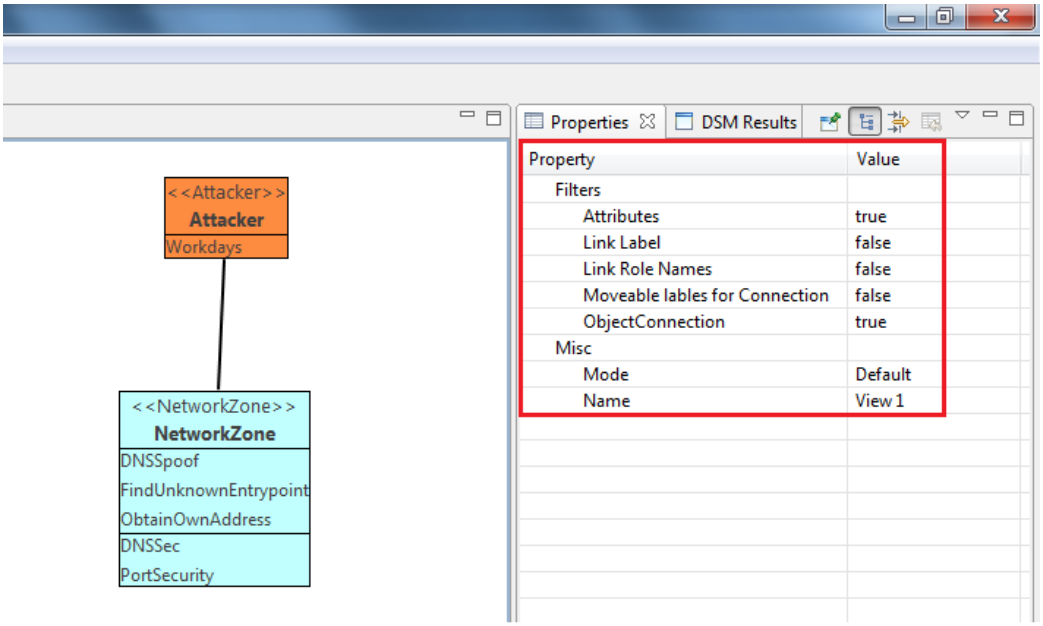

Figure 4: EAAT view properties.

user should select the number of samples that conforms to the time and accuracy required for the analysis. Loosely speaking, more samples means more precise results, but also more time required for the calculation to complete (cf. Section 4.1 for a detailed description). For instance, in Figure 9 a total of 100 samples is chosen (a rather low number). The remainder of the attributes should keep the same values as given in Figure 9.

When a user is satisfied with an object model and has set the configuration parameters for it, all that is left to do to generate analysis results is to press the button “Calculate” (cf. Figure 10).

When calculations are complete results are presented to the user by color-coding all templates and attack steps on a scale from 0%: green - 50%: yellow - 100%: red. Here, probability refers to the likelihood that one or more professional penetration testers are successful with the attack step in the object model within the time designated for the attack. Probabilities for templates are derived based on the estimates for the attack steps associated with it and the currently chosen color profile. An example color profile is “Mean”, which denotes that the color is derived based on the mean likelihood value for all relevant attack steps; another profile is “Access”, which only considers attack steps that result in access

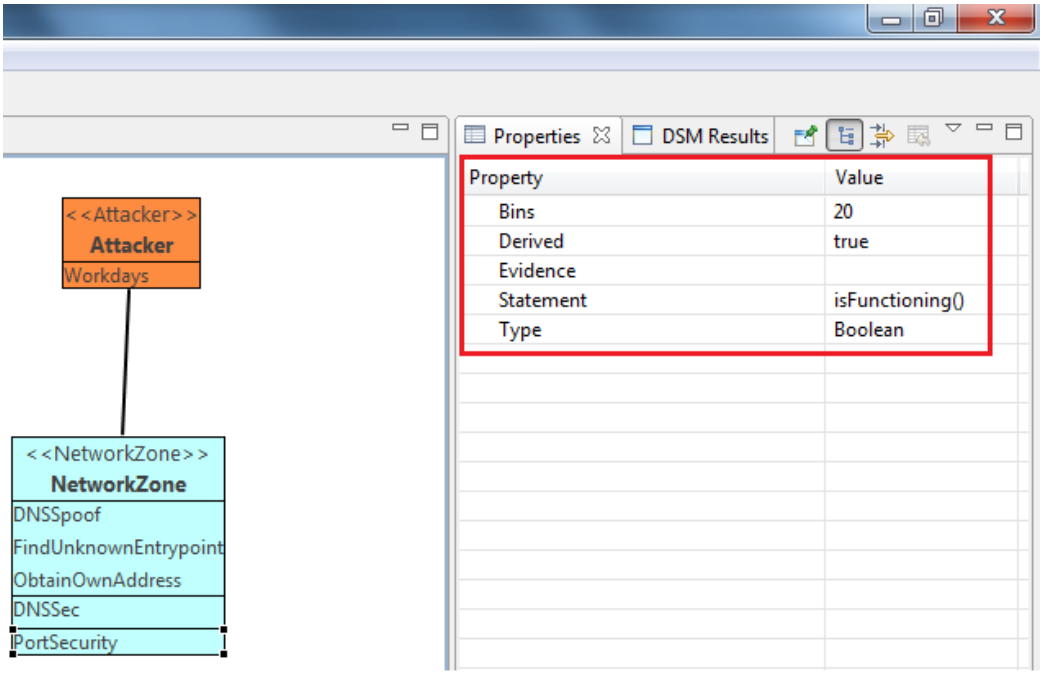

Figure 5: EAAT properties of attack steps and defenses.

of assets (e.g., an operating system). The "Mean" color profile is exemplified in Figure 11 (before calculation) and Figure 12 (after calculation). As can be seen in Figure 11, the attack step `FindUnknownEntryPoint`, that an attacker is able to find a network zone entrypoint unknown to the system administrator, is approximately 69% likely under the very basic conditions of the object model (what these conditions are, and how they relate is the main locus of this report). The overall coloring scheme is visualized in Figure 13.

4. The Cyber Security Modeling Language

4.1. P²AMF logic of CySeMoL

This section describes the specifics of CySeMoL in terms of P²AMF logic. The entire code-base, which consists of 5210 lines of code, is available for download[1].

[1] www.ics.kth.se/eaat/downloads

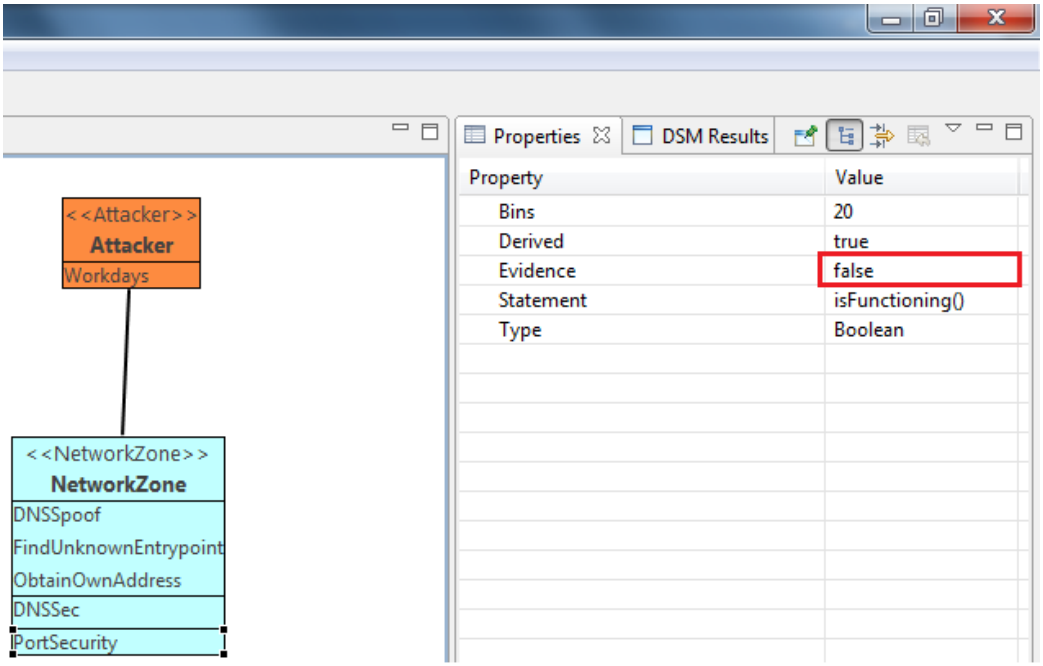

Figure 6: EAAT changing the state of an attack step or defense.

There are four types of concepts in CySeMoL: `Attacker`, `AttackStep`, `Defense` and `Asset` (see Figure 15). Each `AttackStep` and `Defense` is connected to an `Asset` that it compromises or protects. For instance, the `AttackStep` `FindUnknownService` and the `Defense` `AntiMalware` are connected to the `Asset` `OperatingSystem`. The template corresponding to the asset `NetworkZone` can be seen in Figure 14 (both from a meta model view and from an object model view). In an object model, the user is not required to specify these relations - the user simply depicts and connects *templates* related to `Assets`, and any `AttackStep` or `Defense` associated to these are depicted automatically.

Connections between `AttackSteps` are then derived automatically depending on how the user has connected `Assets` in an object model. For example, if a user has connected an `ApplicationServer` to a `NetworkZone`, then there will be a derived connection from the `AttackStep` `NetworkZone.ObtainAddress` to the `AttackStep` `ApplicationServer.ConnectTo`. Any `AttackStep` that an `Attacker` is connected to is considered entry-point for the attack.

OCL operations are used to examine what `AttackSteps` that are reachable (through any means) by one or more `Attackers` from one or more source `AttackSteps`, for each P^{2}AMF sample. CySeMoL thus describes how often

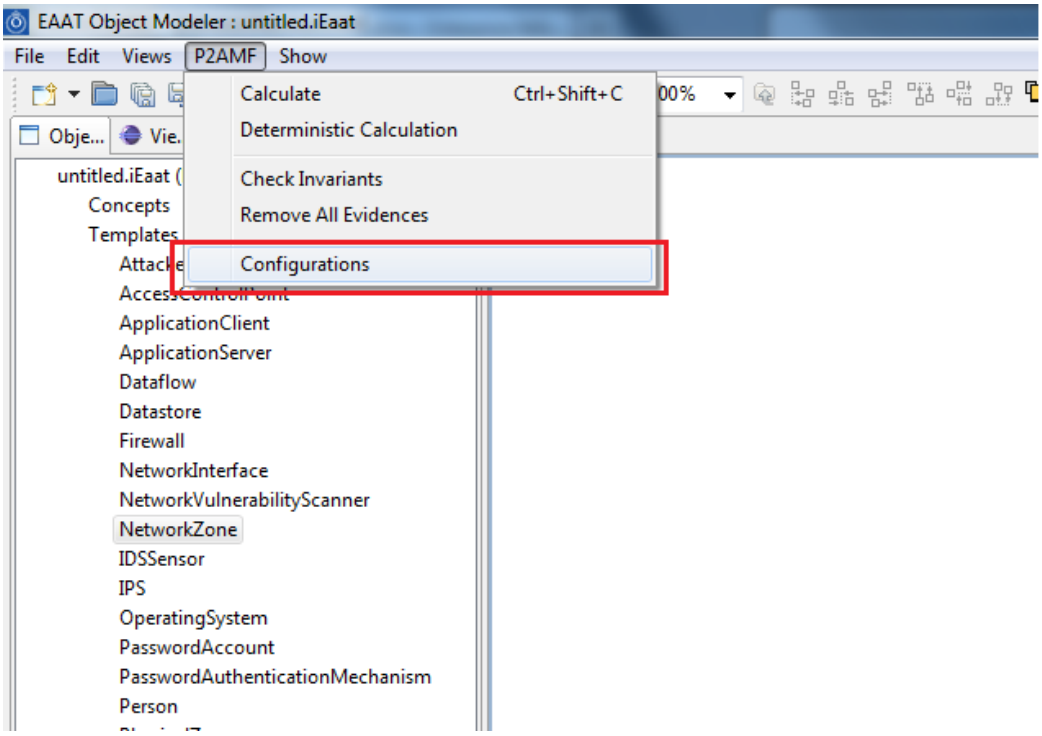

Figure 7: EAAT calculation configuration.

`Attackers` are able to reach different `AttackSteps` in an object model. The primary OCL operation for this purpose is `nextAttackWave`, a recursive function that attempts to visit each `AttackStep` in an object model (the array `visited` keeps track of the ones that have been visited). To visit some `AttackSteps`, there is a need to be able to visit multiple of their parents; this is managed by `unlockedSteps`, which keeps track of `AttackSteps` that have been *unlocked* during the execution of `nextAttackWave`. `unlockedSteps` is required as an `AttackStep` might require an attacker to successfully visit multiple parent attack steps, which in turn might not be possible based on a single model traversal. `getAttackSteps` and `getPaths` serve to check whether it is possible to visit the `AttackSteps` connected to one the currently examined; this is examined by calling `isAccessible`. In other words, the Bayesian logic of `AttackSteps` in CySeMoL is stored in `isAccessible` through Bernoulli distributions and IF/ELSE statements. This is exemplified by the fictive attack graph in Figure 16. In this simple example, the likelihood of successful arbitrary code execution is 21% given a scenario where the attacker is able to connect to a vulnerable software that is

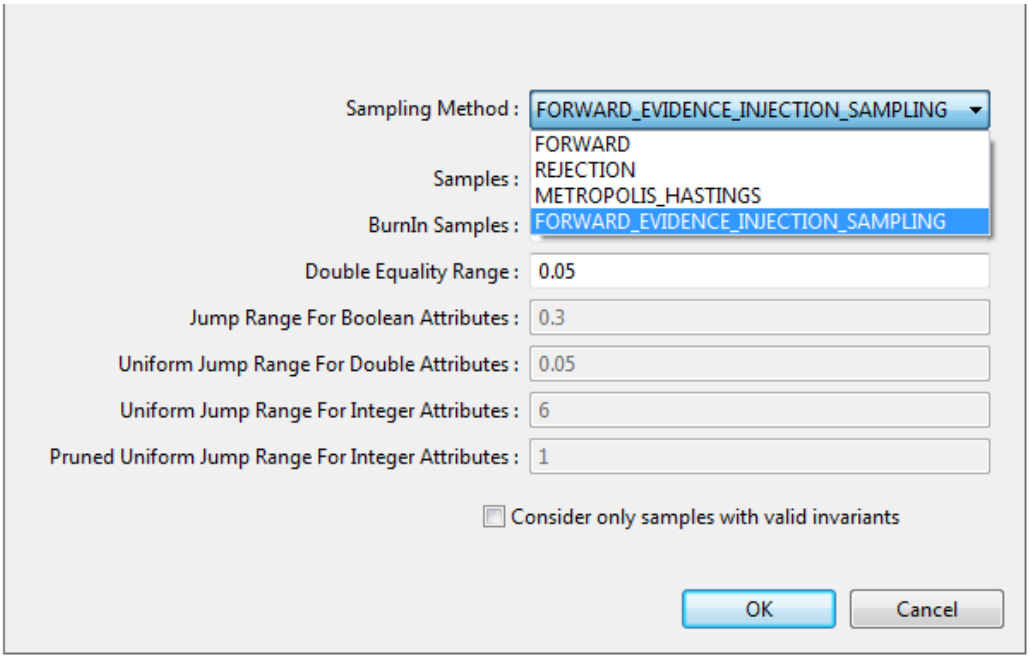

Figure 8: EAAT sampling methods.

protected by a network intrusion detection system and an anti-malware solution.

A small code example of `isAccessible` can be seen for `SocialEngineerCredentials` in Algorithm 1. `SocialEngineerCredentials` is considered visited if two conditions are fulfilled: 1) an `AttackStep` `Interface` must be connected to it and part of `visited`, and 2) the attacker must be able to decieve the individual. If the individual has undergone awareness training (i.e., `AwarenessProgram.Functioning` = TRUE), then the variable `awarenessprogramTrue` must be TRUE; else, `awarenessprogramFalse` must be TRUE. Similarly, the Bayesian logic of each `Defense` is stored in `isFunctioning`. For both `AttackStep` and `Defense`, the operation `defenseAvailable` checks whether any `Defense` relevant for the Bayesian logic in `isAccessible` and `isFunctioning` is available.

`AttackStep.Likelihood` denotes the probability of an attacker being able to successfully utilize a particular `AttackStep`; it is the fraction of samples where that particular `AttackStep` could be visited. `Defense.Functioning` denotes the likelihood that a defense mechanism such as `AwarenessProgram` is available. `Attacker.Time` denotes how much time that the attacker has available for the attack.

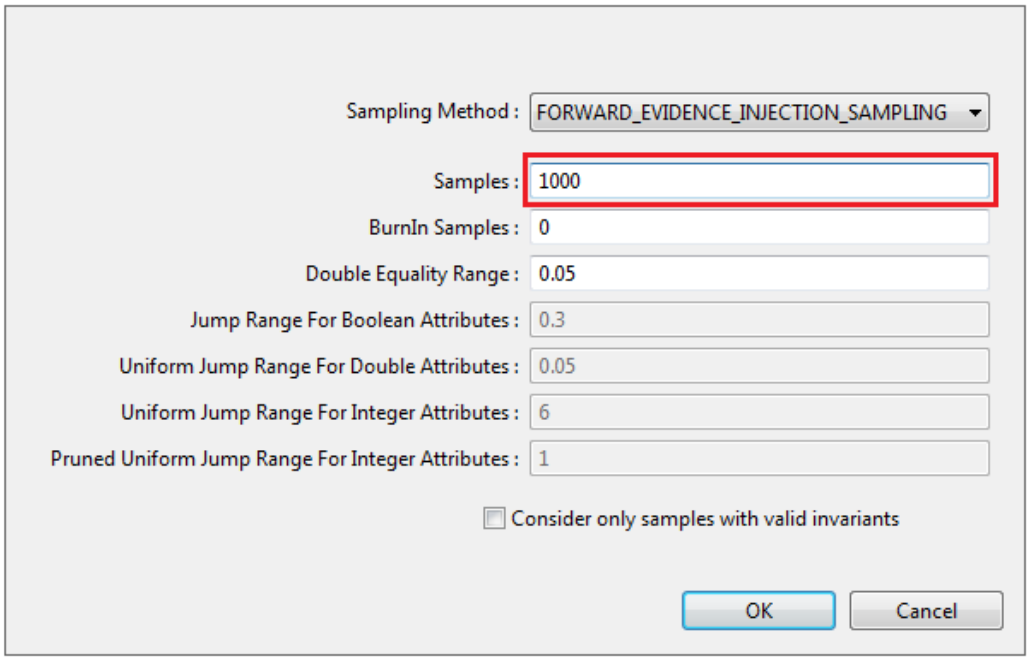

Figure 9: EAAT samples

```
let awarenessprogramTrue : Real = bernoulli(exp(0.0715,Attacker.Time))
let awarenessprogramFalse : Real = bernoulli(exp(0.241,Attacker.Time))
if visited->intersection(self.interface)->notEmpty() then
    if self.person.awarenessprogram.functioning then
        awarenessprogramTrue
    else
        awarenessprogramFalse
    end
else
    False
end
```

Algorithm 1: `SocialEngineerCredentials.isAccessible`

4.2. CySeMoL metamodel

An overview of the CySeMoL metamodel can be seen in Figure 17. Here, each "box" refers to a template that couples assets to defenses and attack steps (cf. Section 3.1 and Section 4.1). In total, CySeMoL consists of 23 assets, 59 attack steps, 58 defenses and 51 relationships between assets. Section 5 -

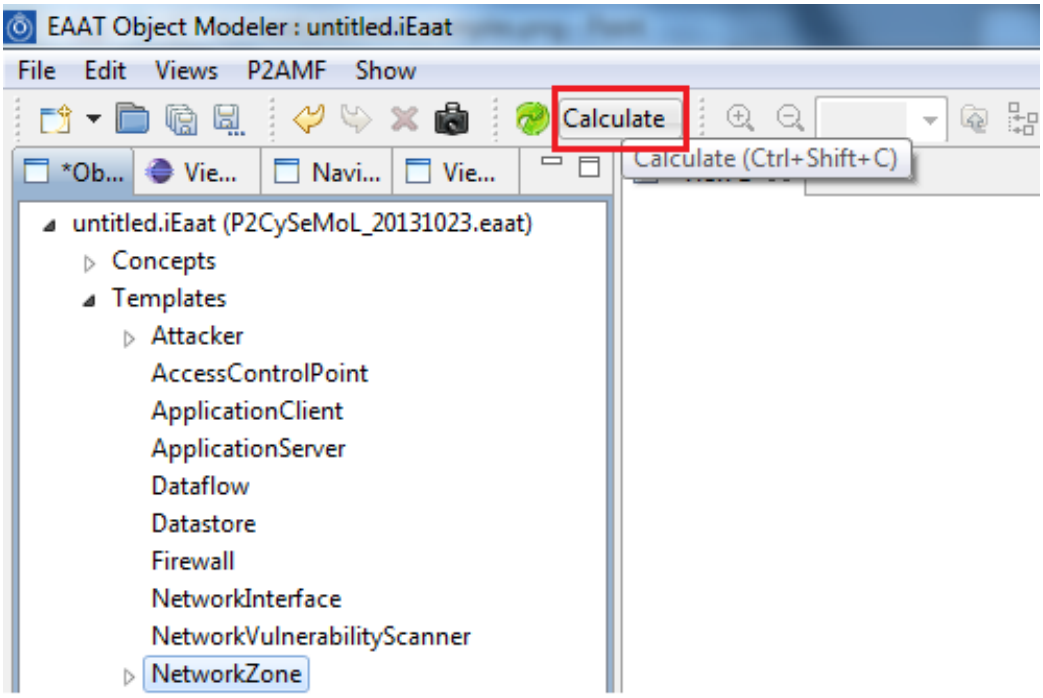

Figure 10: EAAT run calculation.

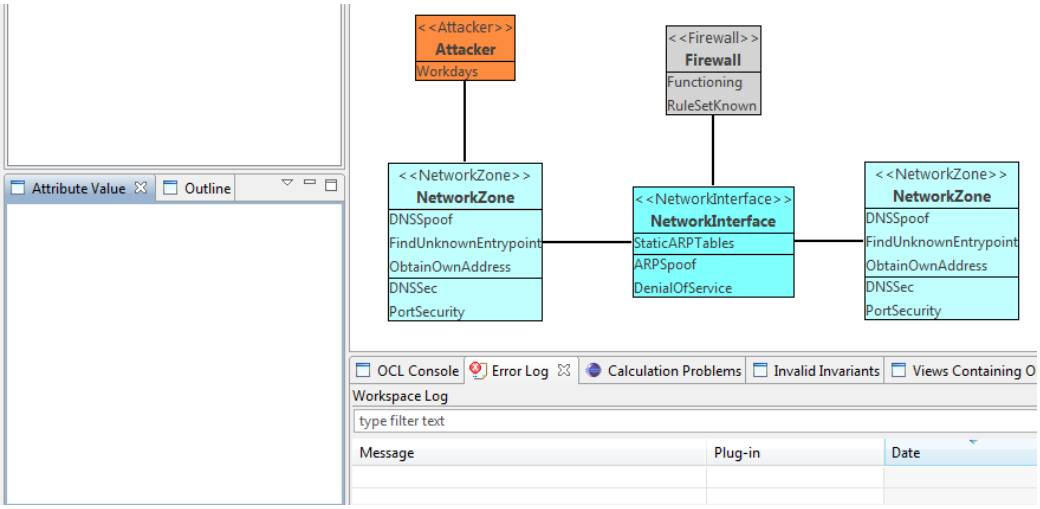

Figure 11: Example model before calculation.

Section 28 describe the templates of CySeMoL in depth. As templates from

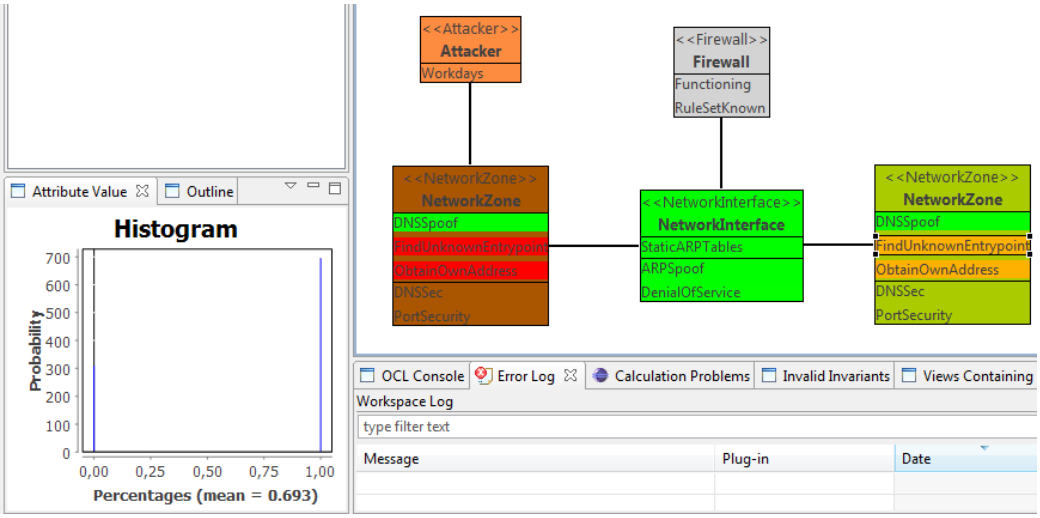

Figure 12: Example model after calculation.

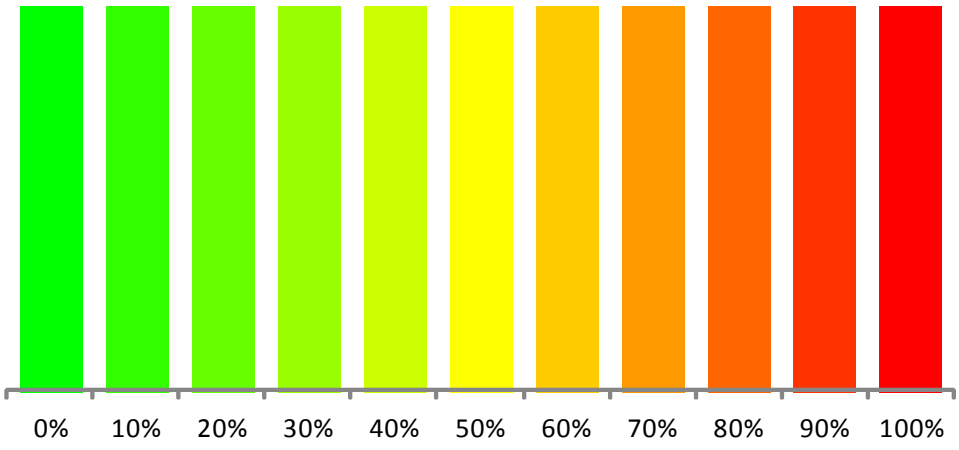

Figure 13: CySeMoL coloring calculation coloring scheme (percentages refer to likelihood of attack success).

a user perspective are the same as assets, but with coupled attack steps and defenses, they are later in this report also referred to simply as *assets*. One template, `Attacker`, is not present in Figure 17 for pedagogical reasons; this

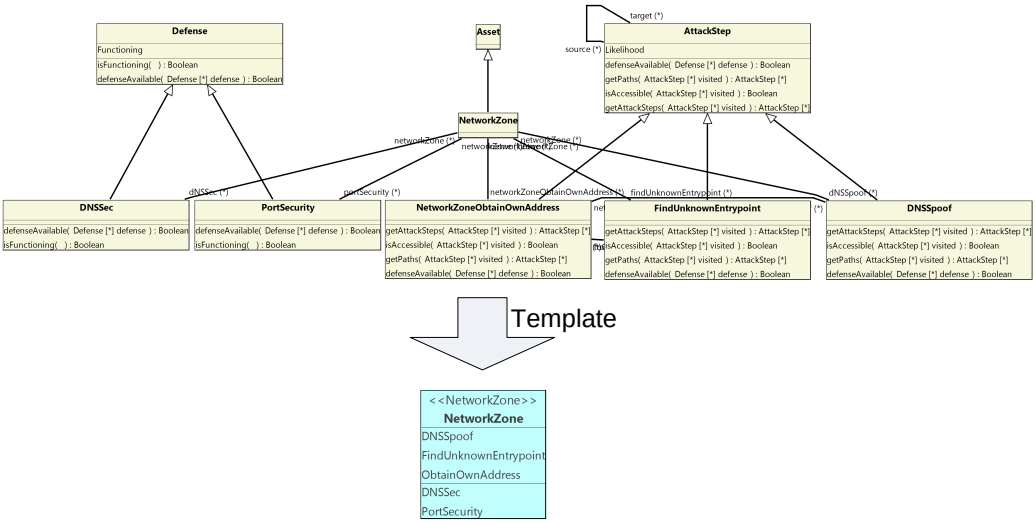

Figure 14: The template corresponding to `NetworkZone` (above meta model viewpoint, below object model viewpoint).

template can be connected to any attack step in CySeMoL. For the same reason, `AttackStep.Likelihood` and `Defense.Functioning` are not shown in Figure 17. When a user changes the state of an attack step or defense, it is `AttackStep.Likelihood` or `Defense.Functioning` that are concerned.

With the exception of `Attacker`, all templates are described in the same means: First, some overall information about the template is outlined. Second, all possible connections for the template are described using both text and a figure. Third, the attack steps and defenses of each template are described through a table. Fourth, the attack steps and defenses corresponding to the template are described in depth. Each attack step and defense is also described in a predictable manner - first, overall information about it is presented, then, the quantitative logic corresponding to it is described.

Furthermore, frequently within these sections, mathematical denotations for distributions are mentioned. For instance, in Algorithm 1 "*bernoulli(exp(0.0715,`Attacker.Time`))*" is denoted. Here, `Attacker.Time` refers to the number of workdays that an attacker is able to spend (cf. Section 5). *exp(0.0715,`Attacker.Time`)* denotes that the probability corresponding to a cumulative density function for an exponential distribution with $\lambda = 0.0715$ and a specific `Attacker.Time` is concerned. *bernoulli()* is used to convert this number into a probability that EAAT and P^{2}AMF are able to parse (a bernoulli distribu-

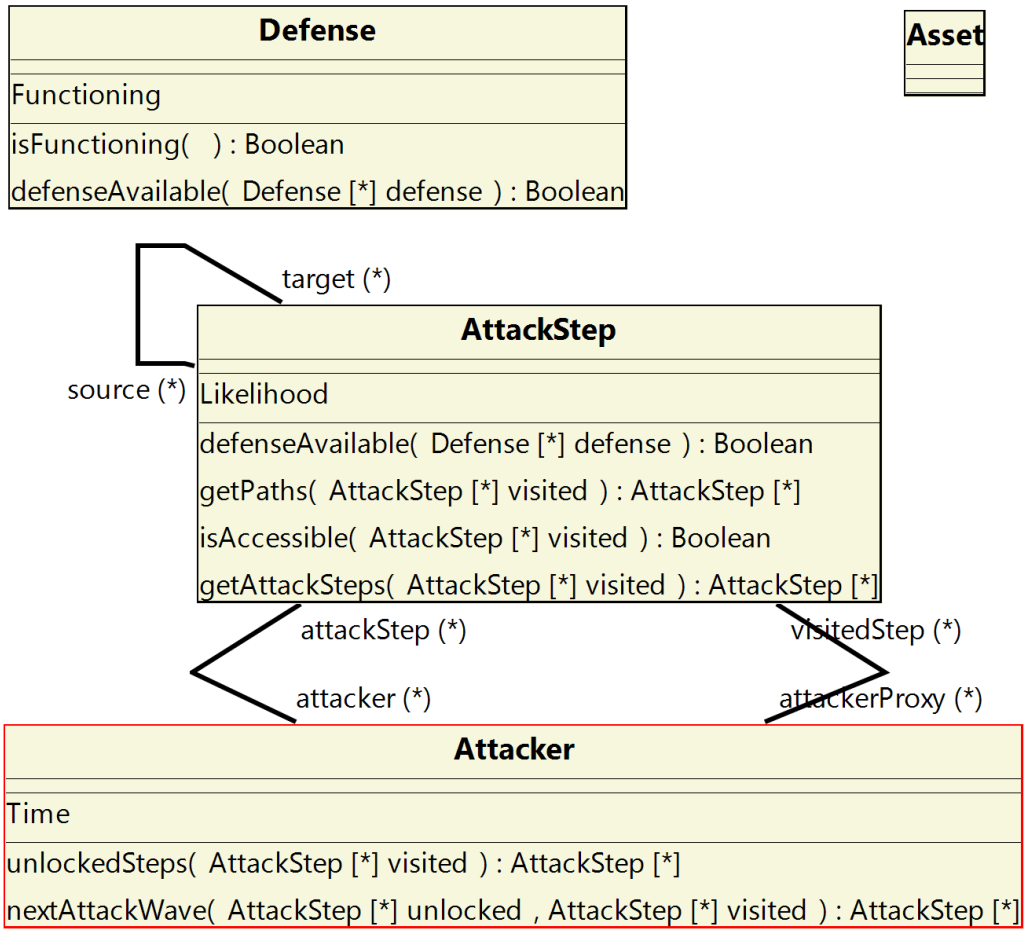

Figure 15: `Attacker`, `AttackStep`, `Defense` and `Asset`. OCL operations are given in the lower box of each class; attributes are given in the upper box of each concept.

tion concerns two states; in the case of CySeMoL, TRUE and FALSE). In total, CySeMoL employ's four different distribution functions:

- Exponential (exp)
- Gamma (gam)
- Log-normal (ln)
- Pareto (par)

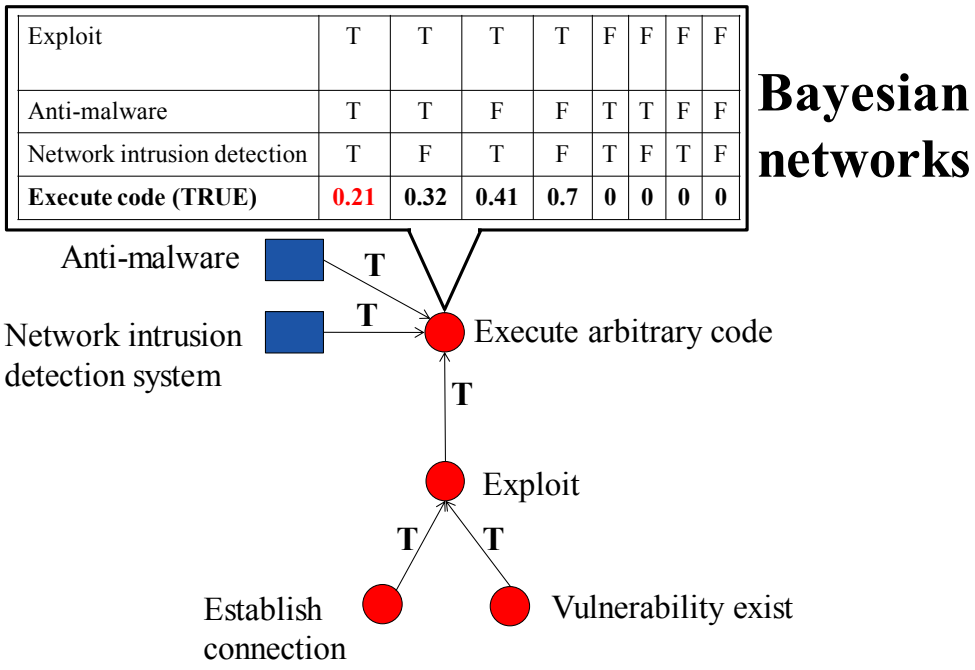

Figure 16: An example attack graph.

In addition to these distribution functions, CySeMoL also employ's linear interpolation. This is denoted as "linear".

5. Attacker

In CySeMoL, an `Attacker` constitutes an individual who is determined to compromise assets of a depicted object model. Naturally, the characteristics of this attacker will influence what attacks that are possible, and how likely different activities are to succeed [13]. In CySeMoL, it is assumed that the attacker is a professional penetration tester with access to publicly available tools and techniques. Consequently, the attack steps and estimates within CySeMoL need to be viewed in the light of this attacker profile.

An `Attacker` can be connected to any class that has an attack step; connecting the attacker to an attack step within a class denotes the source attack vector. This particular attack step always evaluates to TRUE, regardless of the properties of the object model.

The `Attacker` has one attribute - `Time`. This specifies how many workdays an attacker has to spend on each attack step for an object model. Computationally,

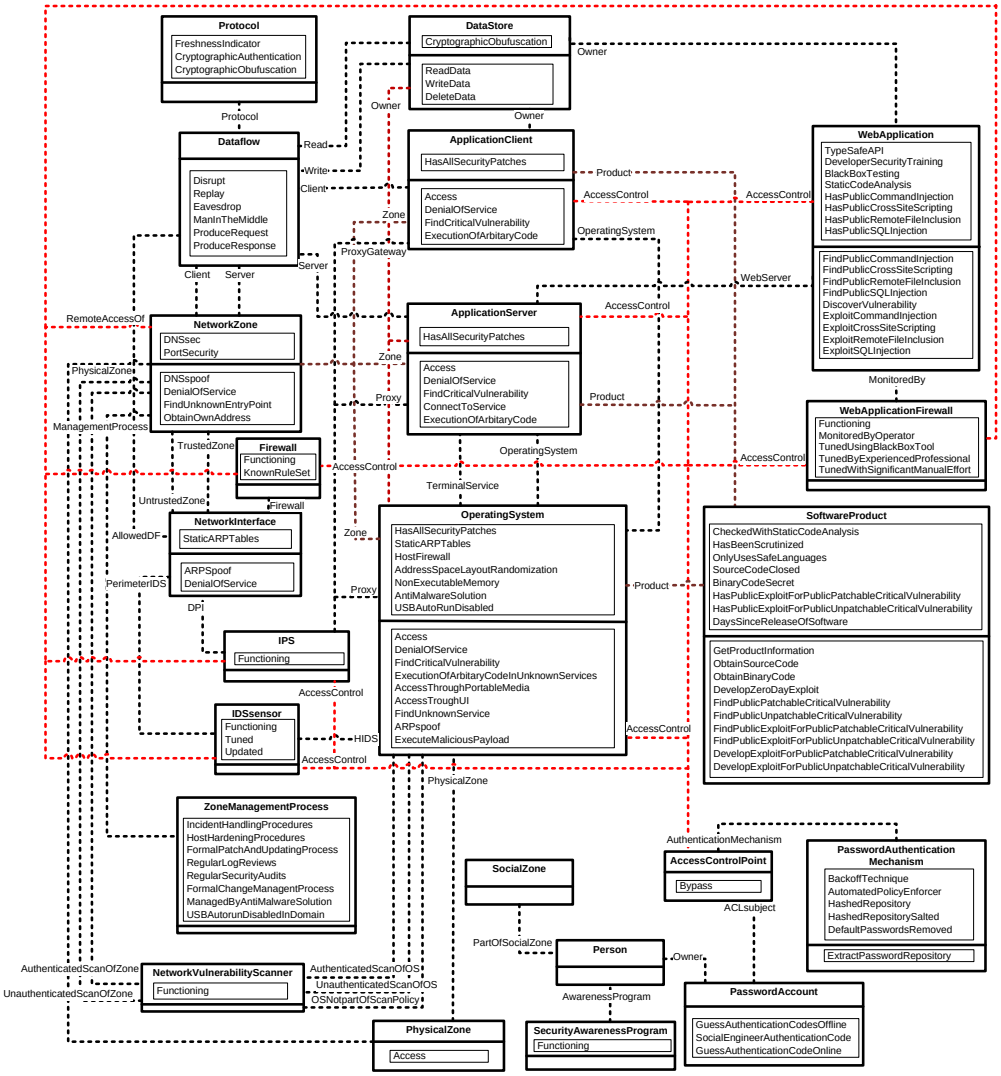

Figure 17: An overview of the CySeMoL metamodel.

the probability of each attack step in an object model being TRUE is evaluated with respect to the number of work days specified for any modeled attackers. For instance, the likelihood of successful social engineering of a security-trained individual is estimated according to: *bernoulli(exp(0.0715,*`Attacker.Time`*))* (cf. Section 25 for details). For this particular attack step, if an attacker has two workdays to spend, the likelihood of success is 13%; given ten days, the likelihood is

51%.

6. Operating System

An `OperatingSystem` (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. CySeMoL makes a clear distinction between software programs and OSs. Software programs are often tightly coupled to an operating system; for instance, Windows environments are tightly coupled to a service message block (SMB) service that, for example, enables file sharing and remote printing. However, in CySeMoL any optional software program that come with operating systems should be represented through the classes `ApplicationClient` and `ApplicationServer`, and only "mandatory" core functionality (e.g. the TCP/IP stack implementation) should be seen as a part of the `OperatingSystem` class. Example OSs include Windows 8, Mac OS X and VMWare VSphere 5. Any OS running inside a hypervizor such as VMWare VSphere 5 would also be considered an `OperatingSystem`.

An `OperatingSystem` can be connected to ten different assets in thirteen different means (see Figure 18). An `ApplicationClient` or `ApplicationServer` should be connected to an OS that enables its execution. Here, an `ApplicationServer` can act as either a terminal to its core services (e.g., telnet, SSH, VNC or RDP) or expose only application-specific functionality (e.g., an HTTP or FTP server). An OS must be connected to a `SoftwareProduct`; this denotes what type of OS that is concerned. For instance, an enterprise might have 400 computers that run the same type of configuration, e.g., Windows XP SP2; however, the actual patch level of these computers likely differ somewhat. This enterprise would model a single `SoftwareProduct` (Windows XP SP2) and connect this to 400 `OperatingSystems`.

Connection to a `NetworkZone` denotes that the OS has an IP address on this network. Connection to `PhysicalZone` means that an attacker has physical access to the machine running the OS. An `AccessControlPoint` depicts the means of logical access of the content of an OS. `DataStore` is some kind of database located on the OS. An OS can be protected by one or more Intrusion Detection Systems (`IDSSensors`) or Intrusion Prevention Systems (`IPSs`). Finally, an OS can be connected to a `NetworkVulnerabilityScanner` (e.g., Nessus or Qualys Guard), denoting either that the OS is analyzed through an authenticated scan, an unauthenticated scan, or not at all (the latter is used when a `NetworkZone` is part of a scanning policy, but a specific system on the zone should not be).

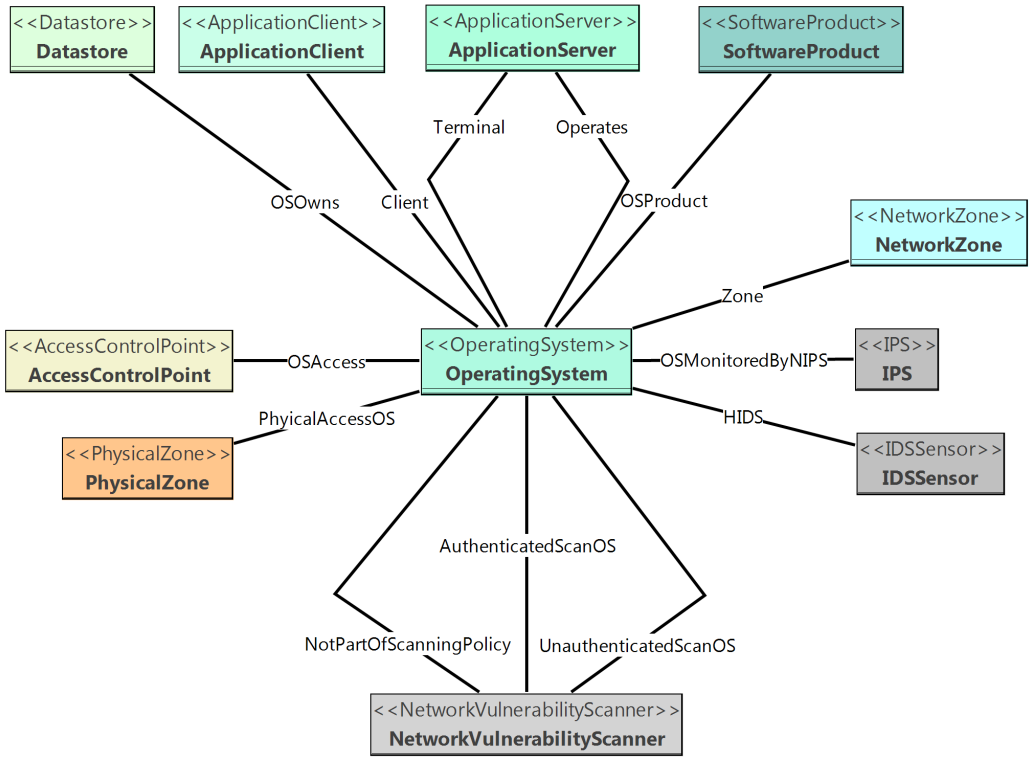

Figure 18: An overview of the connections for `OperatingSystem`.

There are seven defenses and nine attack steps corresponding to OSs; these are shown in Table 1 and desribed in depth in the following sections. The references in Table 1 describe the rationale behind these attributes, e.g., regarding choice of quantitative data.

6.1. Defenses

6.1.1. Has All Security Patches

`HasAllSecurityPatches` denotes whether the OS has had all software security patches implemented [37]. For instance, the Microsoft security update `KB-2861561` address vulnerabilities in Windows Kernel-Mode drivers that could allow remote code execution in various Windows OSs. If a security patch such as `KB-2861561` exist, but is not installed on a vulnerable OS, then `HasAllSecurityPatches` should be FALSE.

The default state of the defense is specified as follows: If an OS is connected to a `NetworkZone`, that in turn is connected to a `ZoneManagementProcess`,

Table 1: Attack steps and defenses of the `OperatingSystem` class.

Attribute	Rationale
Defense	
`HasAllSecurityPatches`	[14, 15]
`StaticARPTables`	[16]
`HostFirewall`	[17]
`AddressSpaceLayoutRandomization`	[18]
`NonExecutableMemory`	[19]
`AntiMalwareSolution`	[20]
`USBAutoRunDisabled`	[21]
Attack step	
`Access`	[22]
`DenialOfService`	[23, 24]
`FindUnknownService`	[25, 26, 27]
`FindCriticalVulnerability`	[28]
`ExecutionOfArbitraryCodeinUnknownService`	[29, 30]
`AccessThroughPortableMedia`	[31]
`AccessThroughUI`	[22]
`ARPSpoof`	[16]
`ExecuteMaliciousPayload`	[32, 33, 34, 35, 36]

then the default state of HasAllSecurityPatches is dependent on the state of ZoneManagementProcess.FormalPatchAndUpdatingProcess (*PM*)(the data for this is gathered from [14]). If an OS, or a NetworkZone connected to the OS, is connected to a NetworkVulnerabilityScanner (and the OS is depicted to be part of the scan), then the default state of the defense is dependent on the type of scan that is designated (authenticated *ANS* or unauthenticated *UNS*; according to the data presented in [15]). If none of these scenarios apply, the default state of HasAllSecurityPatches is FALSE. This logic is summarized in Table 2.

Table 2: Defenses affecting likelihood of HasAllSecurityPatches.

PM	**ANS**	**UNS**	**Data**
TRUE	TRUE	TRUE	bernoulli(0.79)
TRUE	TRUE	FALSE	bernoulli(0.79)
TRUE	FALSE	TRUE	bernoulli(0.79)
TRUE	FALSE	FALSE	bernoulli(0.79)
FALSE	TRUE	TRUE	bernoulli(0.637)
FALSE	TRUE	FALSE	bernoulli(0.637)
FALSE	FALSE	TRUE	bernoulli(0.3028)
FALSE	FALSE	FALSE	0

6.1.2. Static ARP Tables

StaticARPTables involves if the OS has functioning static Address Resolution Protocol (ARP) tables. The ARP tables map logical IP addresses to physical MAC addresses in the broadcast domain of the operating system [16].

The default state of this defense is FALSE; it is not dependent on the existence of any other defense.

6.1.3. Host Firewall

A HostFirewall [17], for instance, the Windows firewall, is assumed to allow all DataFlows from/to the OS and any software employed on it. It serves to block services that are unknown to the modeler (OperatingSystem.FindUnknownService).

The default state of this defense is TRUE; it is not dependent on the existence of any other defense.

6.1.4. Address Space Layout Randomization

The purpose of `AddressSpaceLayoutRandomization` (ASLR) is to introduce randomness into memory addresses used by a given software module [18]. This will make a class of exploit techniques fail with a quantifiable probability and also allow their detection since failed attempts will most likely crash the attacked task. ASLR has for instance been available for Windows OSs since Windows Vista.

The default state of this defense is TRUE as most modern OSs have it implemented; it is not dependent on the existence of any other defense.

6.1.5. Non Executable Memory

`NonExecutableMemory` is a feature which if implemented and functional is intended to prevent an application or service from executing code from a non-executable memory region [19]. If the OS has non-executable memory implemented and working there should thus be a smaller likelihood of success for a certain type of exploits (buffer overflow attacks). An example of this defense mechanism is Data Execution Prevention (DEP), which is available for Microsoft OS from Windows XP SP2 and onward.

The default state of this defense is TRUE as most modern OSs and hardware support it; it is not dependent on the existence of any other defense.

6.1.6. Anti Malware Solution

`AntiMalwareSolution`, or anti-virus, is software used to prevent, detect, remove and report malicious software (i.e., malware) [20]. Many exploits involve injection of some type of software code; an `AntiMalwareSolution` has a chance to detect and prevent such code.

If an OS is connected to a `NetworkZone`, that in turn is connected to a `ZoneManagementProcess`, and `ZoneManagementProcess.ManagedByAntiMalwareSolution` is TRUE, then the default state of `AntiMalwareSolution` is TRUE; it is FALSE in other cases.

6.1.7. USB AutoRun Disabled

`USBAutoRunDisabled` involves whether the autorun functionality (that is enabled per default in most OSs) has been disabled. If it is disabled it will increase the difficulty of propagation for USB-based malware [21, 38, 39].

If an OS is connected to a `NetworkZone`, that in turn is connected to a `ZoneManagementProcess`, then the default state of `USBAutoRunDisabled` is

TRUE if ZoneManagementProcess.USBAutoRunDisabledInDomain is TRUE; it is FALSE in other cases.

6.2. Attack Steps

6.2.1. Access

OperatingSystem.Access denotes whether an attacker is able to manage the content of an OS as an administrator of it. In CySeMoL, there are two means of accomplishing this: either the actor succeeds with OperatingSystem.ExecuteMaliciousPayload or OperatingSystem.AccessThroughUI. The prior involves installation of malware that enables remote access of the OS; the latter involves bypassing the PasswordAuthenticationMechanism for the AccessControlPoint of the OS or an ApplicationServer that acts as a terminal to it.

If any of these attack steps are TRUE, this attack step is TRUE; else, it is FALSE.

6.2.2. Denial of Service

This attack step groups all attacks that target the OS and attempts to cause denial of service (DoS) [23, 24]. In CySeMoL, there are essentially two means of causing DoS for an OS: either by OperatingSystem.Access or OperatingSystem.ExecuteMaliciousPayload.

If any of these attack steps are TRUE, this attack step is TRUE; else, it is FALSE.

6.2.3. Find Unknown Service

If the attacker can find services unknown to the network administrator running on a host it is possible to attack these to gain privileges on it. As unknown services are likely to have more security holes (e.g. as they are not prospect of patch management) they are severe security issues.

Three variables influence OperatingSystem.FindUnknownService: whether the OS has a HostFirewall enabled (*HF*), whether network administrators have performed hardening, e.g., removing unnecessary services (ZoneManagementProcess.HostHardeningProcedures, *HHP*), and whether network administrators have defined a formal change management process (ZoneManagementProcess.FormalChangeManagementProcess, *FCMP*). The estimates corresponding to the likelihood of success for this attack step under these circumstances are described in Table 3; these data come from [25, 26, 27].

Table 3: Defenses affecting likelihood of `FindUnknownService`.

HHP	**FCMP**	**HF**	**Data**
TRUE	TRUE	TRUE	bernoulli(0.595)
TRUE	FALSE	TRUE	bernoulli(0.83)
FALSE	TRUE	TRUE	bernoulli(0.69)
FALSE	FALSE	TRUE	bernoulli(0.899)
TRUE	TRUE	FALSE	bernoulli(0.69)
TRUE	FALSE	FALSE	bernoulli(0.878)
FALSE	TRUE	FALSE	bernoulli(0.69)
FALSE	FALSE	FALSE	bernoulli(0.925)

6.2.4. Find Critical Vulnerability

This attack step involves whether an attacker is able to acquire a critical exploit for a binary service running on the OS [28]. For this to be successful, there is a need for the attacker to find an unknown service on the OS (`OperatingSystem.FindUnknownService`), and an exploit for this service. CySeMoL accounts for five different means for attackers to obtain exploits (all located in `SoftwareProduct`):

1. a public exploit could be released for a patchable vulnerability,
2. an exploit could be developed by the attacker for a patchable vulnerability,
3. a public exploit could be released for a non-patchable vulnerability,
4. an exploit could be developed by the attacker for a non-patchable vulnerability,
5. an exploit could be developed by the attacker for a zero-day vulnerability discovered by the attacker.

In CySeMoL, these correspond to:

1. `FindPublicExploitForPatchableCriticalVulnerability`,
2. `DevelopExploitForPatchableCriticalVulnerability`,
3. `FindPublicExploitForUnpatchableCriticalVulnerability`,
4. `DevelopExploitForUnpatchableCriticalVulnerability`,
5. `DevelopZeroDayExploit`.

What type of exploit that is viable depends on whether the OS `HasAllSecurityPatches` or not (if this is TRUE, then only 3-5 would yield TRUE).

6.2.5. Execution of Arbitrary Code in Unknown Service

If an attacker is able to `FindCriticalVulnerability`, then it can attempt to utilize it in order to redirect the control-flow of the application to some code of the attackers choosing [40].

Three defenses affect the likelihood of `ExecutionOfArbitraryCodeinUnknownService`: `AddressSpaceLayoutRandomization` (ASLR), `NonExecutableMemory` (NX) and `IntrusionPreventionSystems` (IPS). How these affect the likelihood of this attack step being true can be seen in Table 4. These data come from [29, 30].

Table 4: Defenses affecting likelihood of `ExecutionOfArbitraryCodeinUnknownService`.

ASLR	**NX**	**IPS**	**Data**
TRUE	TRUE	TRUE	bernoulli(exp(0.025,`Attacker.Time`))
TRUE	FALSE	TRUE	bernoulli(exp(0.288,`Attacker.Time`))
FALSE	TRUE	TRUE	bernoulli(exp(0.046,`Attacker.Time`))
FALSE	FALSE	TRUE	bernoulli(exp(0.266,`Attacker.Time`))
TRUE	TRUE	FALSE	bernoulli(exp(0.103,`Attacker.Time`))
TRUE	FALSE	FALSE	bernoulli(exp(0.155,`Attacker.Time`))
FALSE	TRUE	FALSE	bernoulli(exp(0.108,`Attacker.Time`))
FALSE	FALSE	FALSE	bernoulli(exp(0.379,`Attacker.Time`))

6.2.6. Access Through Portable Media

This variable specifies the possibility of gaining access to an operating system through the use of portable media, for example through auto run exploits or specifically crafted files. In CySeMoL, `AccessThroughPortableMedia` is FALSE if `USBAutoRunDisabled` is TRUE. It is also dependent on whether the user of the OS is part of a `SocialZone` with other users that have had their computers compromised and might `SharePortableMedia`. If not, then this attack step is FALSE. If `USBAutoRunDisabled` is FALSE, and `SocialZone.SharePortableMedia` is TRUE, then it has the following distribution: bernoulli(exp(0.164,`Attacker.Time`)). This data comes from [31].

6.2.7. Access Through UI

This attack step involves whether an attacker is able to gain access to an OS through some login interface of it. In CySeMoL, this attack step

is TRUE if the attacker is able to physically reach the machine and bypass its local authentication mechanism (`PhysicalZone.Access` = TRUE and `ApplicationControlPoint.Bypass` = TRUE), or of the attacker is able to bypass some remote access mechanism to the OS (e.g., SSH or RDP) (ApplicationServer.Access = TRUE (connected to the OS as a `Termninal`)); it is FALSE in other cases.

6.2.8. ARP spoof

This attribute states if it is possible to poison ARP tables in the operating system. If the operating system's ARP tables are poisoned this can be used to intercept traffic going from the external zone to one of the internal zones [16].

The network interface will use its ARP tables to identify what MAC address an incoming IP package should be forwarded to. If the attacker can compromise the ARP table the threat agent can make these IP packages end up at any other MAC address (e.g. the attacker's own). The threat agent could then change the data before forwarding it to the address designated by the sender. A static ARP table (`StaticARPTables`) is a preventive countermeasure against this attack step; if this defense is TRUE, `ARPSpoof` is FALSE. If static ARP tables are not functioning the attacker can accomplish this attack from any network zone through which the corresponding network interface routes traffic [16] (i.e., it is TRUE).

6.2.9. Execute Malicious Payload

This attack step concerns whether a threat agent is able to execute some piece of malicious software on a system [41]. In CySeMoL, this can be accomplished through eight methods:

1. If the OS has a client application that has been compromised by successful arbitrary code execution.
2. If the OS has a server application that is running a web application which has been exploited by a Command Injection attack.
3. If the OS has a server application that is running a web application which has been exploited by a Remote File Inclusion attack.
4. If the OS has a server application that is running a web application which has been exploited by an SQL Injection attack.
5. If it is possible to execute arbitrary code in an unknown service on the OS.
6. If the OS has a terminal server application that has been compromised by successful arbitrary code execution.

7. If the OS has a non-terminal server application that has been compromised by successful arbitrary code execution.
8. If it is possible to inject malicious code through a portable media.

These scenarios correspond to the following CySeMoL attack steps:

1. `ApplicationClient.ExecutionOfArbitraryCode`
2. `WebApplication.ExploitCommandInjection`
3. `WebApplication.ExploitRemoteFileInclusion`
4. `WebApplication.ExploitSQLInjection`
5. `OperatingSystem.ExecutionOfArbitraryCodeInUnknownServices`
6. `ApplicationServer.ExecutionOfArbitraryCode` (Terminal)
7. `ApplicationServer.ExecutionOfArbitraryCode` (Operates)
8. `OperatingSystem.AccessThroughPortableMedia`

If one or more of these attack steps are TRUE, then there is a likelihood of `ExecuteMaliciousPayload` being TRUE; else, this is FALSE. This likelihood furthermore depends on the presence or absence of five defenses:

1. If the OS has a host based intrusion detection system (HIDS) monitoring it.
2. If the network zone connected to the OS has a network based intrusion detection system (NIDS) monitoring it.
3. If the HIDS and/or NIDS are fully updated.
4. If the HIDS and/or NIDS are well tuned.
5. If the OS has an anti-malware installed

In CySeMoL, these correspond to:

1. `IDS.Functioning (with the relation HIDS)`
2. `IDS.Functioning (with the relation NIDS)`
3. `IDS.Updated`
4. `IDS.Tuned`
5. `OperatingSystem.AntiMalwareSolution`

Table 5 describes the likelihood of `ExecuteMaliciousPayload` being TRUE given that there is a HIDS or NIDS (that are not compromised by the attacker). These data come from [32, 33, 34]. If there is no HIDS or NIDS, or these are compromised, but there is an anti-malware, then the likelihood of `ExecuteMaliciousPayload` is 89.7% (this estimate is independent of the time available to the attacker). The data for this parameter come from [35, 36].

Table 5: Defenses affecting likelihood of `ExecuteMaliciousPayload`.

NIDS	HIDS	Updated	Tuned	Data
TRUE	TRUE	TRUE	TRUE	bernoulli(exp(0.127,`Attacker.Time`))
TRUE	TRUE	TRUE	FALSE	bernoulli(exp(0.157,`Attacker.Time`))
TRUE	TRUE	FALSE	TRUE	bernoulli(exp(0.139,`Attacker.Time`))
TRUE	TRUE	FALSE	FALSE	bernoulli(exp(0.187,`Attacker.Time`))
TRUE	FALSE	TRUE	TRUE	bernoulli(exp(0.163,`Attacker.Time`))
TRUE	FALSE	TRUE	FALSE	bernoulli(exp(0.187,`Attacker.Time`))
TRUE	FALSE	FALSE	TRUE	bernoulli(exp(0.168,`Attacker.Time`))
TRUE	FALSE	FALSE	FALSE	bernoulli(exp(0.178,`Attacker.Time`))
FALSE	TRUE	TRUE	TRUE	bernoulli(exp(0.139,`Attacker.Time`))
FALSE	TRUE	TRUE	FALSE	bernoulli(exp(0.151,`Attacker.Time`))
FALSE	TRUE	FALSE	TRUE	bernoulli(exp(0.001,`Attacker.Time`))
FALSE	TRUE	FALSE	FALSE	bernoulli(exp(0.167,`Attacker.Time`))
FALSE	FALSE	TRUE	TRUE	bernoulli(0.8966153846)
FALSE	FALSE	TRUE	FALSE	bernoulli(0.8966153846)
FALSE	FALSE	FALSE	TRUE	bernoulli(0.8966153846)
FALSE	FALSE	FALSE	FALSE	bernoulli(0.8966153846)

7. Application Client

An ApplicationClient is a software, or part of software, that is directly employed by end-users to perform some type of functionality. For instance, a document reader software such as Adobe Reader, or a web browser software such as Firefox.

An ApplicationClient can be connected to seven different assets, each using a single means of connection (cf. Figure 19). An ApplicationClient should be connected to an OperatingSystem that enables its execution. It should also be connected to a SoftwareProduct; this denotes what type of client that is concerned. For instance, an enterprise might have 400 computers that have the application client Adobe Reader installed; however, the actual patch level of these software installations likely differ somewhat. This enterprise would model a single SoftwareProduct (Adobe Reader) and connect this to 400 ApplicationClients (the actual patch levels of Adobe Reader). A client can be connected to a Datastore that depicts information storage.

Connection to a NetworkZone denotes that the client is a combined client/server solution that is possible to remotely interface without having access to core OS functionality (e.g., a VPN-tunnel service). In other words, "traditional" software that only are accessible given local access of a system, such as a web browser, should *not* be connected directly to network zones. An AccessControlPoint depicts the means of logical access of the content of a client. A client can be protected by one or more Intrusion Prevention Systems (IPSs). Finally, a connection to a Dataflow denotes an information flow between the client and one or more servers.

There are one defense and four attack steps corresponding to application servers; these are shown in Table 6 and desribed in depth in the following sections. The references in Table 6 describe the rationale behind these attributes, e.g., regarding choice of quantitative data.

7.1. Defenses

7.1.1. Has All Security Patches

HasAllSecurityPatches denotes whether the client has *all* applicable software security patches implemented [37]. For instance, the Firefox 25 update address MFSA 2013-101, a memory corruption vulnerability in the "workers" module. If a security patch as Firefox 25 exist, but is not installed, then HasAllSecurityPatches should be FALSE.

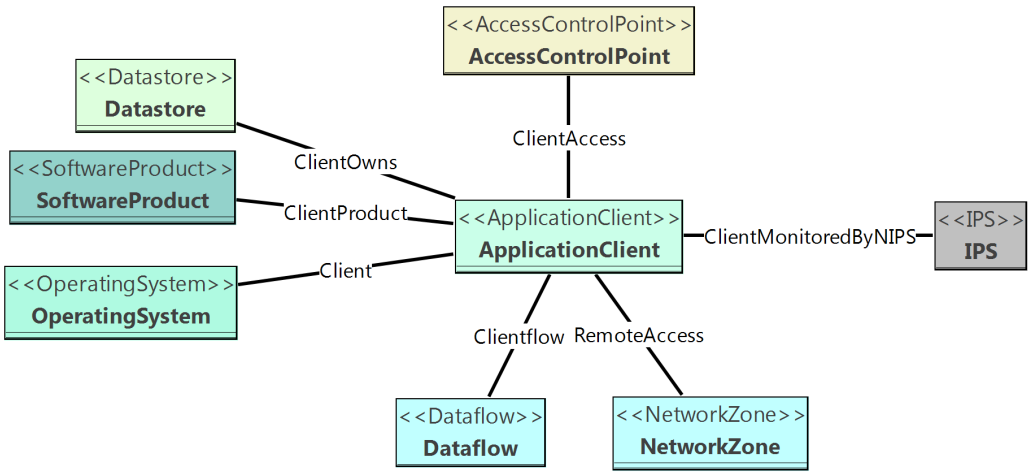

Figure 19: An overview of the connections for `ApplicationClient`.

Table 6: Attack steps and defenses of the `ApplicationClient` class.

Attribute	Rationale
Defense	
`HasAllSecurityPatches`	[14, 15]
Attack step	
`Access`	[22]
`DenialOfService`	[23, 24]
`FindCriticalVulnerability`	[28]
`ExecutionOfArbitraryCode`	[29, 30]

The default state of the defense is specified as follows: If a client is connected to a `NetworkZone` (directly or through an OS), that in turn is connected to a `ZoneManagementProcess`, then the default state of `HasAllSecurityPatches` is dependent on the state of `ZoneManagementProcess.FormalPatchAndUpdatingProcess` (*PM*)(the data for this is gathered from [14]). If the OS running the client, or a `NetworkZone` connected to this OS, is connected to a `NetworkVulnerabilityScanner` (and the OS is depicted to be part of the scan), then the default state of the defense is dependent on the type of scan that is designated (authenticated *ANS* or unauthenticated *UNS*; according to the data presented in [15]). If none of these scenarios apply, the default state of `HasAllSecurityPatches` is FALSE. This logic is sum-

marized in Table 7.

Table 7: Defenses affecting likelihood of `HasAllSecurityPatches`.

PM	ANS	UNS	Data
TRUE	TRUE	TRUE	bernoulli(0.79)
TRUE	TRUE	FALSE	bernoulli(0.79)
TRUE	FALSE	TRUE	bernoulli(0.79)
TRUE	FALSE	FALSE	bernoulli(0.79)
FALSE	TRUE	TRUE	bernoulli(0.637)
FALSE	TRUE	FALSE	bernoulli(0.637)
FALSE	FALSE	TRUE	bernoulli(0.3028)
FALSE	FALSE	FALSE	0

7.2. Attack Steps

7.2.1. Access

`Access` denotes whether an attacker is able to manage the content of a client as an administrator of it. In CySeMoL, success of this attack requires the attacker to circumvent the login function of the client (`AccessControlPoint.Bypass`). There is also a need for the attacker being able to connect to the client itself. In CySeMoL, this can be accomplished through access to the OS that executes the client (`OperatingSystem.Access`), or if the client is connected to a `NetworkZone` reachable by the attacker (i.e., `NetworkZone.ObtainOwnAddress` = TRUE). If so, this attack step is TRUE; else, it is FALSE.

7.2.2. Denial Of Service

This attack step groups all attacks that target the server and attempts to cause denial of service (DoS) [23, 24]. In CySeMoL, this can be accomplished by an attacker that has access to the OS hosting the client. If so, it is TRUE; else, it is FALSE.

7.2.3. Find Critical Vulnerability

This attack step involves whether an attacker is able to acquire a critical exploit for a client [28]. For this to be successful, there is a need for the attacker to be able to interface with the client (`OperatingSystem.Access` or if the client is connected to a `NetworkZone` reachable by the attacker), and find an exploit for

the client. CySeMoL accounts for five different means for attackers to obtain exploits (all located in `SoftwareProduct`):

1. a public exploit could be released for a patchable vulnerability,
2. an exploit could be developed by the attacker for a patchable vulnerability,
3. a public exploit could be released for a non-patchable vulnerability,
4. an exploit could be developed by the attacker for a non-patchable vulnerability,
5. an exploit could be developed by the attacker for a zero-day vulnerability discovered by the attacker.

In CySeMoL, these correspond to:

1. `FindPublicExploitForPatchableCriticalVulnerability`,
2. `DevelopExploitForPatchableCriticalVulnerability`,
3. `FindPublicExploitForUnpatchableCriticalVulnerability`,
4. `DevelopExploitForUnpatchableCriticalVulnerability`,
5. `DevelopZeroDayExploit`.

What type of exploit that is viable depends on whether the server `HasAllSecurityPatches` or not (if this is TRUE, then only 3-5 would yield TRUE).

7.2.4. Execution Of Arbitrary Code

If an attacker is able to `FindCriticalVulnerability`, then it can attempt to utilize it in order to redirect the control-flow of the application to some code of the attackers choosing [40].

Three defenses affect the likelihood of `ExecutionOfArbitraryCode`: `AddressSpaceLayoutRandomization` (ASLR), `NonExecutableMemory` (NX) and `IntrusionPreventionSystems` (IPS). How these affect the likelihood of this attack step being true can be seen in Table 11. These data come from [29, 30].

8. Application Server

An `ApplicationServer` is a software, or part of software, that responds to remote (e.g., over TCP/IP or a serial connection) requests by software clients. For instance, a File Transfer Protocol (FTP) server that provides clients with data stored on directiories shared by the server.

Table 8: Defenses affecting likelihood of `ExecutionOfArbitraryCode`.

ASLR	NX	IPS	Data
TRUE	TRUE	TRUE	bernoulli(exp(0.025,`Attacker.Time`))
TRUE	FALSE	TRUE	bernoulli(exp(0.288,`Attacker.Time`))
FALSE	TRUE	TRUE	bernoulli(exp(0.046,`Attacker.Time`))
FALSE	FALSE	TRUE	bernoulli(exp(0.266,`Attacker.Time`))
TRUE	TRUE	FALSE	bernoulli(exp(0.103,`Attacker.Time`))
TRUE	FALSE	FALSE	bernoulli(exp(0.155,`Attacker.Time`))
FALSE	TRUE	FALSE	bernoulli(exp(0.108,`Attacker.Time`))
FALSE	FALSE	FALSE	bernoulli(exp(0.379,`Attacker.Time`))

An `ApplicationServer` can be connected to eight different assets in ten different ways (cf. Figure 20). It should be connected to an `OperatingSystem` that enables its execution. Depending on the type of this connection, the server either acts as a *Terminal* to access to the core functionality of the OS (e.g., an RDP connection), or as a server that merely *Operates* using the OS to enable its execution. It should also be connected to a `SoftwareProduct`; this denotes what type of server that is concerned. For instance, an enterprise might have 400 computers that run the same type of server, e.g., the server-part of Skype; however, the actual patch level of these software installations likely differ somewhat. This enterprise would model a single `SoftwareProduct` (Skype) and connect this to 400 `ApplicationServers` (the actual patch levels of Skype). A connection to `WebApplication` denotes that the `ApplicationServer` is a web server. A server can be connected to a `Datastore` that depicts information storage.

Connection to a `NetworkZone` denotes that the server is possible to interface by systems on that network. An `AccessControlPoint` depicts the means of logical access of the content of a server. A server can be protected by one or more Intrusion Prevention Systems (`IPSs`). Finally, a connection to a `Dataflow` denotes an information flow between the server and one or more clients.

There are one defense and five attack steps corresponding to application servers; these are shown in Table 9 and desribed in depth in the following sections. The references in Table 9 describe the rationale behind these attributes, e.g., regarding choice of quantitative data.

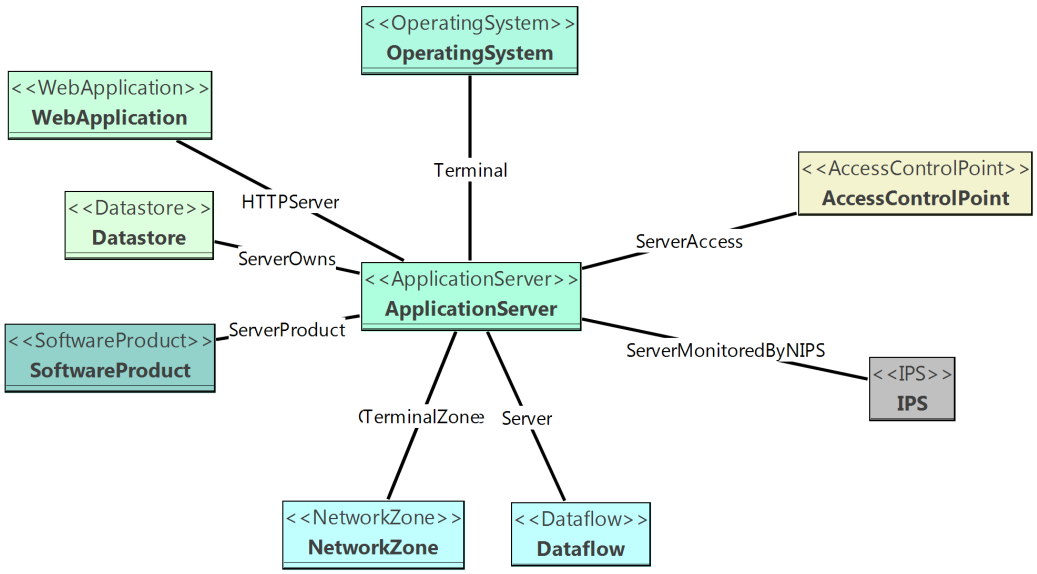

Figure 20: An overview of the connections for `ApplicationServer`.

8.1. Defenses

8.1.1. Has All Security Patches

`HasAllSecurityPatches` denotes whether the server has *all* applicable software security patches implemented [37]. For instance, the Apache httpd server update 2.2.22 address CVE-2012-4557, a denial of service vulnerability in the function `mod_proxy_ajp`. If a security patch as this exist, but is not installed, then `HasAllSecurityPatches` should be FALSE.

The default state of the defense is specified as follows: If a server is connected to a `NetworkZone` (directly or through an OS), that in turn is connected to a `ZoneManagementProcess`, then the default state of `HasAllSecurityPatches` is dependent on the state of `ZoneManagementProcess.FormalPatchAndUpdatingProcess` (*PM*)(the data for this is gathered from [14]). If the OS running the server, or a `NetworkZone` connected to this OS, is connected to a `NetworkVulnerabilityScanner` (and the OS is depicted to be part of the scan), then the default state of the defense is dependent on the type of scan that is designated (authenticated *ANS* or unauthenticated *UNS*; according to the data presented in [15]). If none of these scenarios apply, the default state of `HasAllSecurityPatches` is FALSE. This logic is summarized in Table 10.

Table 9: Attack steps and defenses of the `ApplicationServer` class.

Attribute	**Rationale**
Defense	
`HasAllSecurityPatches`	[14, 15]
Attack step	
`ConnectToServer`	[22]
`Access`	[22]
`DenialOfService`	[23, 24]
`FindCriticalVulnerability`	[28]
`ExecutionOfArbitraryCode`	[29, 30]

Table 10: Defenses affecting likelihood of `HasAllSecurityPatches`.

PM	**ANS**	**UNS**	**Data**
TRUE	TRUE	TRUE	bernoulli(0.79)
TRUE	TRUE	FALSE	bernoulli(0.79)
TRUE	FALSE	TRUE	bernoulli(0.79)
TRUE	FALSE	FALSE	bernoulli(0.79)
FALSE	TRUE	TRUE	bernoulli(0.637)
FALSE	TRUE	FALSE	bernoulli(0.637)
FALSE	FALSE	TRUE	bernoulli(0.3028)
FALSE	FALSE	FALSE	0

8.2. Attack Steps

8.2.1. Connect To Server

This attack step concerns whether an attacker is able to connect to the server software itself. In CySeMoL, this is possible if the attacker can reach a dataflow, network zone or OS that the server is connected to. If this can be accomplished, it is TRUE; else, it is FALSE.

8.2.2. Access

`Access` denotes whether an attacker is able to manage the content of a server as an administrator of it. In CySeMoL, this can be accomplished by circumventing the login function of the server (`AccessControlPoint.Bypass`). There is also a need for the attacker being able to connect to the server itself (`ConnectToServer`). If both of these attack steps are TRUE, this attack step is TRUE; else, it is FALSE.

8.2.3. Denial Of Service

This attack step groups all attacks that target the server and attempts to cause denial of service (DoS) [23]. In CySeMoL, it is enough that an attacker can succeed with the attack step `ConnectToServer` for this attack step to be TRUE (else, it is FALSE). This was discovered through a survey involving evaluating the significance of variables commonly thought to incluence DoS of server software [24].

8.2.4. Find Critical Vulnerability

This attack step involves whether an attacker is able to acquire a critical exploit for a server [28]. For this to be successful, there is a need for the attacker to be able to connect to the server (`ConnectToServer`), and find an exploit for the server. CySeMoL accounts for five different means for attackers to obtain exploits (all located in `SoftwareProduct`):

1. a public exploit could be released for a patchable vulnerability,
2. an exploit could be developed by the attacker for a patchable vulnerability,
3. a public exploit could be released for a non-patchable vulnerability,
4. an exploit could be developed by the attacker for a non-patchable vulnerability,
5. an exploit could be developed by the attacker for a zero-day vulnerability discovered by the attacker.

In CySeMoL, these correspond to:

1. FindPublicExploitForPatchableCriticalVulnerability,
2. DevelopExploitForPatchableCriticalVulnerability,
3. FindPublicExploitForUnpatchableCriticalVulnerability,
4. DevelopExploitForUnpatchableCriticalVulnerability,
5. DevelopZeroDayExploit.

What type of exploit that is viable depends on whether the server HasAllSecurityPatches or not (if this is TRUE, then only 3-5 would yield TRUE).

8.2.5. Execution Of Arbitrary Code

If an attacker is able to FindCriticalVulnerability, then it can attempt to utilize it in order to redirect the control-flow of the application to some code of the attackers choosing [40].

Three defenses affect the likelihood of ExecutionOfArbitraryCode: AddressSpaceLayoutRandomization (ASLR), NonExecutableMemory (NX) and IntrusionPreventionSystems (IPS). How these affect the likelihood of this attack step being true can be seen in Table 11. These data come from [29, 30].

Table 11: Defenses affecting likelihood of ExecutionOfArbitraryCode.

ASLR	NX	IPS	Data
TRUE	TRUE	TRUE	bernoulli(exp(0.025,Attacker.Time))
TRUE	FALSE	TRUE	bernoulli(exp(0.288,Attacker.Time))
FALSE	TRUE	TRUE	bernoulli(exp(0.046,Attacker.Time))
FALSE	FALSE	TRUE	bernoulli(exp(0.266,Attacker.Time))
TRUE	TRUE	FALSE	bernoulli(exp(0.103,Attacker.Time))
TRUE	FALSE	FALSE	bernoulli(exp(0.155,Attacker.Time))
FALSE	TRUE	FALSE	bernoulli(exp(0.108,Attacker.Time))
FALSE	FALSE	FALSE	bernoulli(exp(0.379,Attacker.Time))

9. Software Product

A SoftwareProduct represents a software that has not yet been altered or updated with patches. For example, in an enterprise there could be hundreds of computers using the software product Linux Red Hat 4 and a hundred computers using

Windows XP Service Pack 1. In this case Linux Red Hat 4 and Windows XP Service pack are two `SoftwareProducts` and their hundred installations are of the class `OperatingSystem`. A `SoftwareProduct` can be connected to three assets, each by a single type of connection: `OperatingSystem`, `ApplicationClient`, and `ApplicationServer` (see Figure 21).

The distinction between software product and software instance is made since the flaws of a software installation to great extent are associated to the product it is an installation of. For instance, Windows XP SP1 will have some flaws associated to it and Linux Red Hat 4 will have other. These are inherent and constitute a potential vulnerability in all installations of this product. The installations can however apply patches and updates to eliminate them. It should be noted that some vulnerabilities are introduced by updates and patches. This relationship is not considered in CySeMoL. CySeMoL instead assigns general probabilities to attack steps that identify exploits against the software product. Exploits are classified in three dimensions: their severity, if there is a patch available for the vulnerability that the attacker may exploit, and if the exploit code or clear instructions is readily available in the public domain (e.g. posted on a website). Table 19 describes the three dimensions.

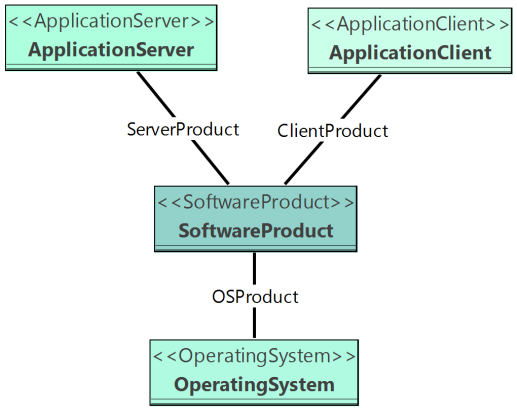

Figure 21: An overview of the connections for `SoftwareProduct`.

There are seven defenses and eight attack steps corresponding to software products; these are shown in Table 12 and desribed in depth in the following sections. The references in Table 12 describe the rationale behind these attributes, e.g., regarding choice of quantitative data.

Table 12: Attack steps and defenses of the `SoftwareProduct` class.

Attribute	Rationale
Defense	
`SourceCodeSecret`	[42]
`BinarySecret`	[42]
`ImprovedWithStaticCodeAnalysis`	[43]
`WrittenOnlyInSafeLanguages`	[19]
`HasBeenScrutinized`	[44]
`HasNoUnpatchableVulnerability`	[45]
`HasNoPatchableVulnerability`	[45]
Attack step	
`GetProductInformation`	[46]
`FindPublicPatchableCriticalVulnerability`	[45]
`FindPublicUnpatchableCriticalVulnerability`	[45]
`FindPublicExploitForPatchableCriticalVulnerability`	[45]
`DevelopExploitForPatchableCriticalVulnerability`	[45]
`FindPublicExploitForUnpatchableCriticalVulnerability`	[45]
`DevelopExploitForUnpatchableCriticalVulnerability`	[45]
`DevelopZeroDayExploit`	[47]

9.1. Defenses

9.1.1. Source Code Secret

If the attacker gains access to the software product source code it is possible to search it and "white box" test it for vulnerabilities [42].

Some software is open source, in which case it is easy to obtain the source code. Other software is proprietary and closed; obtaining the source code is then more difficult. CySeMoL does not decompose the steps that could be taken to obtain the source code of a product; this could however be introduced. As it is now, the probability is used to express how difficult this attack step is to succeed given such variables marginalized. It is instead recommended that users set the state of this attack step based on the assumptions used in the analysis.

The default state of this variable is TRUE as most commercial software are closed-source.

9.1.2. Binary Secret

If an attacker have access to the binary code (machine code) it is possible to conduct "black box" tests and thereby detect vulnerabilities [42]. If the attacker cannot obtain this code (not even by compiling the soruce code) it will be almost impossible to find a new vulnerability in th software.

If the software is open, e.g. as freeware then it is easy to gain access to the binary. If the software is closed or a custom made product then it might be extremely difficult to obtain a binary. CySeMoL does not decompose the steps that could be taken to obtain the binary code to a product; this could however be introduced. As it is now, the probability is used to express how difficult this attack step is to succeed given such variables marginalized.

The default state of this variable is FALSE as very few software are difficult to obtain.

9.1.3. Improved With Static Code Analysis

A static code analyser is a tool that inspects the source code of software to find bugs or vulnerabilities in it [43]. The idea goes back decades and has contemporary tools are recognized as effective. To apply such tools is recommended best practice. It is for example included in Microsoft's secure development lifecycle. Twenty of the thirty enterprises involved in BSIMM (Building Security In Maturity Model) use automated tools for code analysis.

The default state of this variable is TRUE as most commercial software regularly undergo static code analysis.

9.1.4. Written Only In Safe Languages

If a "safe" programming language like Java or Python, which performs boundary checking, have been used then the possibilities of performing buffer overflows is reduced and thereby also the opportunity of finding a vulnerability. If an "unsafe" language like C or C++ has been used the likelihood of finding a vulnerability is greater [19].

The use of a safe dialect of these languages is however possible, e.g. Cyclone. In that case it is seen as a safe language. The use of safe libraries to embed the unsafe code is also included in this definition. Libsafe and libverify are examples of safe libraries for C/C++.

The default state of this variable is FALSE as most commercial software are written in "unsafe" languages.

9.1.5. Has Been Scrutinized

Some software products have been scrutinized. That is, they have been thoroughly tested for vulnerabilities. Research has shown that the frequency of vulnerability discovery in software products decrease over time [44].

The default state of this variable is TRUE as most commercial software regularly are tested for security vulnerabilities by both researchers and practitioners.

9.1.6. Has No Public Patchable Vulnerability

This concerns a scenario where it is known that the software has no patchable vulnerability available on any public forums such as the NVD, PacketStorm or Exploit DB.

The default state of this variable is FALSE.

9.1.7. Has No Public Unpatchable Vulnerability

This concerns a scenario where it is known that the software has no unpatchable vulnerability available on any public forums such as the National Vulnerability Database (NVD), PacketStorm or Exploit DB.

The default state of this variable is FALSE.

9.2. Attack Steps

9.2.1. Get Product Information

This attack step concerns whether an attacker is able to successfully probe a software to determine its properties. For instance, that a discovered FTP server actually is IIS FTP 7.5.

To reach this attack step, there is a need for the attacker to either be able to:

1. produce a response from a client software connected to the product,
2. connect to a server software connected to the product,
3. find an unknown software on an OS connected to the product.

These three scenarios correspond to the following CySeMoL concepts:

1. `ApplicationClient.ProduceResponse`,
2. `ApplicationServer.ConnectTo`,
3. `OperatingSystem.FindUnknownService`.

If any of these three attack steps are TRUE, then the success rate of this attack step is 99% (else, it is FALSE). This data comes from a study of how frequently network vulnerability scanners correctly identifies software products [46].

9.2.2. Find Public Patchable Critical Vulnerability

This attack step concerns whether an attacker is able to find a patchable critical vulnerability on some public domain. To reach this attack step, there is a need for the corresponding `GetProductInformation` to be TRUE. If so, the likelihood of success depends on whether the software is an OS (`OperatingSystem`) or not (`ApplicationClient` or `ApplicationServer`) (cf. Table 13). These data come from [45].

Table 13: Data on `FindPublicPatchableVulnerability`.

Software type	**Data**
OS	bernoulli(gamma(0.014, 3630.1, `Attacker.Time`))
Application	bernoulli(par(29.06, 0, `Attacker.Time`))

Finally, if the defense `HasNoPublicPatchableVulnerability` is set to TRUE, then this attack step is FALSE.

9.2.3. Find Public Unpatchable Critical Vulnerability

This attack step concerns whether an attacker is able to find a patchable critical vulnerability on some public domain. To reach this attack step, there is a need for the corresponding `GetProductInformation` to be TRUE. If so, the likelihood of success depends on whether the software is an OS (`OperatingSystem`) or not (`ApplicationClient` or `ApplicationServer`) (cf. Table 14). These data come from [45].

Finally, if the defense `HasNoPublicUnpatchableVulnerability` is set to TRUE, then this attack step is FALSE.

Table 14: Data on `FindPublicUnpatchableVulnerability`.

Software type	Data
OS	bernoulli(lognormal(4.85, 0.98, `Attacker.Time`))
Application	bernoulli(gamma(0.036, 7755.7, `Attacker.Time`))

9.2.4. Find Public Exploit For Patchable Critical Vulnerability

This attack step concerns whether an attacker is able to find an exploit for a patchable critical vulnerability on some public domain. To reach this attack step, there is a need for `FindPublicUnpatchableVulnerability` or `FindPublicPatchableVulnerability` to be TRUE. That is, a vulnerability that was unpatchable upon its disclosure might be patchable before an exploit is publicly released. If both of these are FALSE, then this attack step is FALSE.

If one, or both, are TRUE then the likelihood of success depends on whether the targeted software is an OS (`OperatingSystem`) or not (`ApplicationClient` or `ApplicationServer`) (cf. Table 15). These data come from [45].

Table 15: Data on `FindPublicExploitForPatchableCriticalVulnerability`.

Software type	Patch	Data
OS	Yes	bernoulli(gamma(0.039, 567.91, `Attacker.Time`))
OS	No	bernoulli(exp(0.0085, `Attacker.Time`))
Application	Yes	bernoulli(gamma(0.033, 1313.37, `Attacker.Time`))
Application	No	bernoulli(exp(0.0734, `Attacker.Time`))

Apart from the time between vulnerabilities, there is also a need to consider the likelihood that the vulnerability of the next exploit released for an OS or application is patchable. Based on the findings of [45], the prior has a likelihood of 9.1%, and the latter 14.5%.

The resulting probabilities for this attack step are thus:

- bernoulli(0.091) * bernoulli(gamma(0.039, 567.91, `Attacker.Time`))
- bernoulli(0.091) * bernoulli(exp(0.0085, `Attacker.Time`))
- bernoulli(0.145) * bernoulli(gamma(0.033, 1313.37, `Attacker.Time`))
- bernoulli(0.145) * bernoulli(exp(0.0734, `Attacker.Time`))

9.2.5. Develop Exploit For Patchable Critical Vulnerability

If a vulnerability is disclosed, but no exploit is made available, then a proficient attacker can utilize the information made public about the vulnerability and devise an exploit for it himself/herself.

To reach this attack step, there is firstly a need for `FindPublicPatchableVulnerability` to be TRUE; else it is FALSE. If TRUE, then the the likelihood of success depends on whether the application's or the patch's source code is available to the attacker (`SourceCodeClosed`). Availability of source code can aid an attacker at discovering "deep" vulnerabilities that are difficult to find using a debugger; it can however also aid defenders at quickly mitigating flaws [48].

The likelihood estimates of this attack step come from [45] and can be seen in Table 16. The estimates are dependent on three assumptions: 1) there is no exploit code (or parts of it) available for the targeted vulnerability, 2) the vulnerable software is not written in a script language, and 3) there is as much information available about the vulnerability as is typical.

Table 16: Data on `DevelopExploitForPatchableCriticalVulnerability`.

`SourceCodeClosed`	**Data**
Yes	bernoulli(linear([0,1,2,5,5.263158], [0,0.05,0.5,0.95,1], Attacker.Time))
No	bernoulli(linear([0,2,5,8,8.421053], [0,0.05,0.5,0.95,1], Attacker.Time))

9.2.6. Find Public Exploit For Unpatchable Critical Vulnerability

This attack step concerns whether an attacker is able to find an exploit for an unpatchable critical vulnerability on some public domain. To reach this attack step, there is a need for `FindPublicUnpatchableVulnerability` to be TRUE (if it is FALSE, then this attack step is FALSE).

If `FindPublicUnpatchableVulnerability` is TRUE then the likelihood of success depends on whether the targeted software is an OS (`OperatingSystem`) or not (`ApplicationClient` or `ApplicationServer`) (cf. Table 17). These data come from [45].

Apart from the time between vulnerabilities, there is also a need to consider the likelihood that the vulnerability of the next exploit released for an OS or application is unpatchable. Based on the findings of [45], the prior has a likelihood of 90.9%, and the latter 85.5%. The resulting probabilities for this attack step are thus:

Table 17: Data on FindPublicExploitForUnpatchableCriticalVulnerability.

Software type	Data
OS	bernoulli(lognormal(-49.39, 26.93, Attacker.Time))
Application	bernoulli(lognormal(-52.71, 22.85, Attacker.Time))

- bernoulli(0.909) * bernoulli(lognormal(-49.39, 26.93, Attacker.Time))
- bernoulli(0.855) * bernoulli(lognormal(-52.71, 22.85, Attacker.Time))

9.2.7. Develop Exploit For Unpatchable Critical Vulnerability

If an unpatchable vulnerability is disclosed, but no exploit is made available, then a proficient attacker can utilize the information made public about the vulnerability and devise an exploit for it himself/herself. The difference from the attack step DevelopExploitForPatchableCriticalVulnerability is that the present concerns a vulnerability for which there is no released software patch mitigating it available.

To reach this attack step, there is firstly a need for FindPublicUnpatchableVulnerability to be TRUE; else it is FALSE. If TRUE, then the the likelihood of success depends on whether the application's or the patch's source code is available to the attacker (SourceCodeClosed). Availability of source code can aid an attacker at discovering "deep" vulnerabilities that are difficult to find using a debugger; it can however also aid defenders at quickly mitigating flaws [48].

The likelihood estimates of this attack step come from [45] and can be seen in Table 18. The estimates are dependent on three assumptions: 1) there is no exploit code (or parts of it) available for the targeted vulnerability, 2) the vulnerable software is not written in a script language, and 3) there is as much information available about the vulnerability as is typical.

Table 18: Data on DevelopExploitForUnpatchableCriticalVulnerability.

SourceCodeClosed	Data
Yes	bernoulli(linear([0,1,3,6,6.315789], [0,0.05,0.5,0.95,1], Attacker.Time))
No	bernoulli(linear([0,4,7,11,11.57895], [0,0.05,0.5,0.95,1], Attacker.Time))

9.2.8. Develop Zero Day Exploit

If there is no disclosed vulnerability available, or if the attacker wants to exploit something that is unknown to be vulnerable in the public domain, then (s)he can attempt to discover a new vulnerability (often called a zero-day).

To reach this attack step, there is a need for the corresponding `GetProductInformation` to be TRUE (else, it is FALSE). If so, then the probability of the attack step being TRUE depends on the presence or absence of four variables:

- if the targeted software has been scrutinized before,
- whether the attacker has access to the application source code,
- whether the software is written using a (memory) "safe" language, dialect or library,
- if the software has been analyzed by static code analyzers and improved based on the result.

In CySeMoL, these scenarios correspond to:

- `SoftwareProduct.HasBeenScrutinized`,
- `SoftwareProduct.SourceCodeSecret`,
- `SoftwareProduct.WrittenOnlyInSafeLanguages`,
- `SoftwareProduct.ImprovedWithStaticCodeAnalysis`.

For each scenario, it is assumed that the attacker is able to study the application binary (e.g., using a debugger). I.e., `BinarySecret` = TRUE. If `BinarySecret` is FALSE, then this attack step is also FALSE.

The likelihood estimates for the 16 scenarios corresponding to these four defenses come from [47] and can be seen in Table 19.

10. Web Application

A `WebApplication` (WA) is a software that is run on a HTTP(S) server. Typical WAs have both client-side and server-side functionality. Client-side functionality typically concern features that are intepreted by web browsers (e.g., Firefox

Table 19: Data on `DevelopZeroDayExploit`.

Project	Scrutinized	SourceCode	SafeLanguage	CodeAnalyzers	Data
1	Yes	Yes	Yes	Yes	bernoulli(linear([0,3,13,74,77.89474], [0,0.05,0.5,0.95,1], Attacker.Time))
2	Yes	Yes	Yes	No	bernoulli(linear([0,1,6,27,28.42105], [0,0.05,0.5,0.95,1], Attacker.Time))
3	Yes	Yes	No	Yes	bernoulli(linear([0,1,3,26,27.36842], [0,0.05,0.5,0.95,1], Attacker.Time))
4	Yes	Yes	No	No	bernoulli(linear([0,0.001,4,9,9.473684], [0,0.05,0.5,0.95,1], Attacker.Time))
5	Yes	No	Yes	Yes	bernoulli(linear([0,0.001,13,26,27.36842], [0,0.05,0.5,0.95,1], Attacker.Time))
6	Yes	No	Yes	No	bernoulli(linear([0,0.001,3,17,17.89474], [0,0.05,0.5,0.95,1], Attacker.Time))
7	Yes	No	No	Yes	bernoulli(linear([0,0.001,1,7,7.368421], [0,0.05,0.5,0.95,1], Attacker.Time))
8	Yes	No	No	No	bernoulli(linear([0,1,3,8,8.421053], [0,0.05,0.5,0.95,1], Attacker.Time))
9	No	Yes	Yes	Yes	bernoulli(linear([0,1,12,855,900], [0,0.05,0.5,0.95,1], Attacker.Time))
10	No	Yes	Yes	No	bernoulli(linear([0,1,14,344,362.1053], [0,0.05,0.5,0.95,1], Attacker.Time))
11	No	Yes	No	Yes	bernoulli(linear([0,0.001,10,27,28.42105], [0,0.05,0.5,0.95,1], Attacker.Time))
12	No	Yes	No	No	bernoulli(linear([0,1,4,27,28.42105], [0,0.05,0.5,0.95,1], Attacker.Time))
13	No	No	Yes	Yes	bernoulli(linear([0,2,9,855,900], [0,0.05,0.5,0.95,1], Attacker.Time))
14	No	No	Yes	No	bernoulli(linear([0,1,6,18,18.94737], [0,0.05,0.5,0.95,1], Attacker.Time))
15	No	No	No	Yes	bernoulli(linear([0,1,4,257,270.5263], [0,0.05,0.5,0.95,1], Attacker.Time))
16	No	No	No	No	bernoulli(linear([0,0.001,3,9,9.473684], [0,0.05,0.5,0.95,1], Attacker.Time))

or Internet Explorer) and their resources (e.g., an adobe flash plugin). Server-side functionality concerns functions that are executed in the context environment of the server software. This includes, for instance, PHP and ASP.NET applications. In CySeMoL, a WA is considered to contain a combination of both server-side and client-side application functions as this typically is the case in practice.

The reason to why there is a specific asset concerning WAs, rather than simply modeling them as `ApplicationServers`, is because the vulnerability discovery process is radically different for compiled and script-based software.

A WA can be connected to three different assets (see Figure 22). An `ApplicationServer` need be connected to it to designate that this `ApplicationServer` is a web server exposing the WA to an external environment. A `WebApplicationFirewall` (WAF) is a security tool that can be used to prevent attacks against WAs in operation. If this connection is available, the WA is considered to be under the protection of this WAF. A WA can also be connected to a `Datastore` to depict a WA database, e.g., an SQL database.

To model connections to a WA, the user should depict connections to the `ApplicationServer` running the WA. A successful connection to the server denotes a successful connection to the WA.

There are eight defenses and nine attack steps corresponding to web applica-

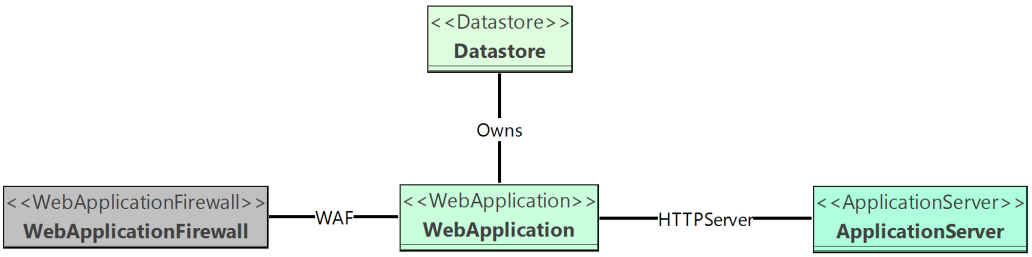

Figure 22: An overview of the connections for `WebApplication`.

tions; these are shown in Table 20 and desribed in depth in the following sections. The references in Table 20 describe the rationale behind these attributes, e.g., regarding choice of quantitative data. The reason for including these four particular classes of vulnerabilities is because they provide radically different outcomes in terms of related attack steps (yet the possibility of exploitation is based on the same dataset [49]).

10.1. Defenses

10.1.1. Type Safe API

Type-safe API's [50] involves using a development environment that is built to function in a secure and reliable fashion. In essence, this countermeasure defines a rule set for allowed code and how different parts of an application exchange information. For instance, how a PHP application is allowed to communicate with an SQL database. If a developer writes code that does not comply with the rule set defined within the type-safe API an error is produced, notifying the developer of the proper syntax as defined by the API.

The default state of this variable is FALSE; this was elicited during a qualitative study on the topic [51].

10.1.2. Developer Security Training

Developer security training [52] involves increasing the WA security awareness of the software developers. The aim is to make developers recognize what improper input and output sanitizing can result in and how such issues can be mitigated.

The default state of this variable is FALSE; this was elicited during a qualitative study on the topic [51].

Table 20: Attack steps and defenses of the `WebApplication` class.

Attribute	Rationale
Defense	
`TypeSafeAPI`	[50, 51]
`DeveloperSecurityTraining`	[52, 51]
`BlackBoxTesting`	[53, 51]
`StaticCodeAnalysis`	[53, 51]
`HasPublicCommandInjectionVulnerability`	[54, 51]
`HasPublicCrossSiteScriptingVulnerability`	[54, 51]
`HasPublicRemoteFileInclusionVulnerability`	[54, 51]
`HasPublicSQLInjectionVulnerability`	[54, 51]
Attack step	
`FindPublicCommandInjectionVulnerability`	[54, 51]
`FindPublicCrossSiteScriptingVulnerability`	[54, 51]
`FindPublicRemoteFileInclusionVulnerability`	[54, 51]
`FindPublicSQLInjectionVulnerability`	[54, 51]
`DiscoverVulnerability`	[55]
`ExploitCommandInjectionVulnerability`	[49]
`ExploitCrossSiteScriptingVulnerability`	[49]
`ExploitRemoteFileInclusionVulnerability`	[49]
`ExploitSQLInjectionVulnerability`	[49]

10.1.3. Black Box Testing

Black box testing [53] involves running automated scanners or fuzzers on deployed WAs without viewing server-side source code. The aim of a black box tests is to find vulnerabilities so that these can be removed before deployment.

The default state of this variable is TRUE; this was elicited during a qualitative study on the topic [51].

10.1.4. Static Code Analysis

Static code analysis [53] involves white box testing for detecting vulnerabilities. They analyze the WA's source code and try to find vulnerabilities that would be exploitable in runtime by applying various checks.

The default state of this variable is TRUE; this was elicited during a qualitative study on the topic [51].

10.1.5. Has Public Command Injection Vulnerability

A command injection vulnerability allows an attacker to execute system level commands [54]. This defense concerns whether there are any command injection exploits available for the WA on public forums such as Exploit DB or PacketStorm. If so, then the state of this defense should be TRUE.

The default state of this variable is FALSE; this was elicited during a qualitative study on the topic [51].

10.1.6. Has Public Cross Site Scripting Vulnerability

A cross site scripting (XSS) vulnerability allows attackers to inject client-side scripts into web pages viewed by other users [54]. This defense concerns whether there are any XSS exploits available for the WA on public forums such as Exploit DB or PacketStorm. If so, then the state of this defense should be TRUE.

The default state of this variable is FALSE; this was elicited during a qualitative study on the topic [51].

10.1.7. Has Public Remote File Inclusion Vulnerability

A remote file inclusion (RFI) vulnerability allows attackers to include remote files in communication with a WA; thus, possible resulting in remote code execution in the context of the web server [54]. This defense concerns whether there are any RFI exploits available for the WA on public forums such as Exploit DB or PacketStorm. If so, then the state of this defense should be TRUE.

The default state of this variable is FALSE; this was elicited during a qualitative study on the topic [51].

10.1.8. Has Public SQL Injection Vulnerability

An SQL injection (SQLi) vulnerability allows attackers to send malicious database queries to an SQL server connected to the WA [54]. This can not only result in an attacker reading or altering content in the SQL database, but also remote code execution. This defense concerns whether there are any SQLi exploits available for the WA on public forums such as Exploit DB or PacketStorm. If so, then the state of this defense should be TRUE.

The default state of this variable is FALSE; this was elicited during a qualitative study on the topic [51].

10.2. Attack Steps

10.2.1. Exploit Command Injection Vulnerability

This attack step concerns whether an attacker is able to successfully exploit a command injection vulnerability in a WA. To reach this attack step, there is a need for `DiscoverVulnerability` or `FindPublicCommandInjection` to be TRUE; else if it FALSE. If one of these attack steps are TRUE, then the likelihood of success depends on the presence and configuration of a `WebApplicationFirewall` protecting the WA. If no WAF is present, then this attack step is TRUE. Else, the likelihood of success depends on the configuration of the WAF. In CySeMoL, the effectiveness of a WAF depends on the four variables:

1. `WebApplicationFirewall.MonitoredByOperator` (OPERATOR)
2. `WebApplicationFirewall.TunedUsingBlackBoxTool` (BBT)
3. `WebApplicationFirewall.TunedByExperiencedProfessional` (EXPERIENCE)
4. `WebApplicationFirewall.TunedWithSignificantManualEffort` (EFFORT)

How these influence the likelihood of attack success is shown in Table 21. These data come from [49].

10.2.2. Exploit Cross Site Scripting Vulnerability

This attack step concerns whether an attacker is able to successfully exploit an XSS vulnerability in a WA. To reach this attack step, there is a need for `DiscoverVulnerability` or `FindPublicCrossSiteScripting` to be TRUE; else if it FALSE. If one of these attack steps are TRUE, then the likelihood of success depends on the presence and configuration of a `WebApplicationFirewall` protecting the WA. If no WAF is present, then this attack step is TRUE. Else, the

Table 21: Data on `ExploitCommandInjectionVulnerability`.

Scenario	OPERATOR	BBT	EXPERIENCE	EFFORT	Data
1	Yes	Yes	Yes	Yes	bernoulli(exp(0.058,Attacker.Time))
2	Yes	Yes	Yes	No	bernoulli(exp(0.126,Attacker.Time))
3	Yes	Yes	No	Yes	bernoulli(exp(0.126,Attacker.Time))
4	Yes	Yes	No	No	bernoulli(exp(0.167,Attacker.Time))
5	Yes	No	Yes	Yes	bernoulli(exp(0.120,Attacker.Time))
6	Yes	No	Yes	No	bernoulli(exp(0.192,Attacker.Time))
7	Yes	No	No	Yes	bernoulli(exp(0.175,Attacker.Time))
8	Yes	No	No	No	bernoulli(exp(0.229,Attacker.Time))
9	No	Yes	Yes	Yes	bernoulli(exp(0.066,Attacker.Time))
10	No	Yes	Yes	No	bernoulli(exp(0.112,Attacker.Time))
11	No	Yes	No	Yes	bernoulli(exp(0.146,Attacker.Time))
12	No	Yes	No	No	bernoulli(exp(0.192,Attacker.Time))
13	No	No	Yes	Yes	bernoulli(exp(0.133,Attacker.Time))
14	No	No	Yes	No	bernoulli(exp(0.192,Attacker.Time))
15	No	No	No	Yes	bernoulli(exp(0.167,Attacker.Time))
16	No	No	No	No	bernoulli(exp(0.263,Attacker.Time))

likelihood of success depends on the configuration of the WAF. In CySeMoL, the effectiveness of a WAF depends on the four variables:

1. `WebApplicationFirewall.MonitoredByOperator` (OPERATOR)
2. `WebApplicationFirewall.TunedUsingBlackBoxTool` (BBT)
3. `WebApplicationFirewall.TunedByExperiencedProfessional` (EXPERIENCE)
4. `WebApplicationFirewall.TunedWithSignificantManualEffort` (EFFORT)

How these influence the likelihood of attack success is shown in Table 22. These data come from [49].

Table 22: Data on `ExploitCrossSiteScriptingVulnerability`.

Scenario	OPERATOR	BBT	EXPERIENCE	EFFORT	Data
1	Yes	Yes	Yes	Yes	bernoulli(exp(0.058,Attacker.Time))
2	Yes	Yes	Yes	No	bernoulli(exp(0.126,Attacker.Time))
3	Yes	Yes	No	Yes	bernoulli(exp(0.126,Attacker.Time))
4	Yes	Yes	No	No	bernoulli(exp(0.167,Attacker.Time))
5	Yes	No	Yes	Yes	bernoulli(exp(0.120,Attacker.Time))
6	Yes	No	Yes	No	bernoulli(exp(0.192,Attacker.Time))
7	Yes	No	No	Yes	bernoulli(exp(0.175,Attacker.Time))
8	Yes	No	No	No	bernoulli(exp(0.229,Attacker.Time))
9	No	Yes	Yes	Yes	bernoulli(exp(0.066,Attacker.Time))
10	No	Yes	Yes	No	bernoulli(exp(0.112,Attacker.Time))
11	No	Yes	No	Yes	bernoulli(exp(0.146,Attacker.Time))
12	No	Yes	No	No	bernoulli(exp(0.192,Attacker.Time))
13	No	No	Yes	Yes	bernoulli(exp(0.133,Attacker.Time))
14	No	No	Yes	No	bernoulli(exp(0.192,Attacker.Time))
15	No	No	No	Yes	bernoulli(exp(0.167,Attacker.Time))
16	No	No	No	No	bernoulli(exp(0.263,Attacker.Time))

10.2.3. Exploit Remote File Inclusion Vulnerability

This attack step concerns whether an attacker is able to successfully exploit a RFI in a WA. To reach this attack step, there is a need for `DiscoverVulnerability` or `FindPublicRemoteFileInclusion` to be TRUE; else if it FALSE. If one of these attack steps are TRUE, then the likelihood of success depends on the presence and configuration of a `WebApplicationFirewall` protecting the WA. If no WAF is present, then this attack step is TRUE. Else, the likelihood of success depends on the configuration of the WAF. In CySeMoL, the effectiveness of a WAF depends on the four variables:

1. `WebApplicationFirewall.MonitoredByOperator` (OPERATOR)
2. `WebApplicationFirewall.TunedUsingBlackBoxTool` (BBT)
3. `WebApplicationFirewall.TunedByExperiencedProfessional` (EXPERIENCE)
4. `WebApplicationFirewall.TunedWithSignificantManualEffort` (EFFORT)

How these influence the likelihood of attack success is shown in Table 23. These data come from [49].

10.2.4. Exploit SQL Injection Vulnerability

This attack step concerns whether an attacker is able to successfully exploit an SQLi vulnerability in a WA. To reach this attack step, there is a need for `DiscoverVulnerability` or `FindPublicSQLInjection` to be TRUE; else if it FALSE. There is also a for a `Datastore` being connected to the WA. If this is the case, then the likelihood of success depends on the presence and configuration of a `WebApplicationFirewall` protecting the WA. If no WAF is present, then this attack step is TRUE. Else, the likelihood of success depends on the configuration of the WAF. In CySeMoL, the effectiveness of a WAF depends on the four variables:

1. `WebApplicationFirewall.MonitoredByOperator` (OPERATOR)
2. `WebApplicationFirewall.TunedUsingBlackBoxTool` (BBT)
3. `WebApplicationFirewall.TunedByExperiencedProfessional` (EXPERIENCE)
4. `WebApplicationFirewall.TunedWithSignificantManualEffort` (EFFORT)

How these influence the likelihood of attack success is shown in Table 24. These data come from [49].

Table 23: Data on `ExploitRemoteFileInclusionVulnerability`.

Scenario	OPERATOR	BBT	EXPERIENCE	EFFORT	Data
1	Yes	Yes	Yes	Yes	bernoulli(exp(0.058,Attacker.Time))
2	Yes	Yes	Yes	No	bernoulli(exp(0.126,Attacker.Time))
3	Yes	Yes	No	Yes	bernoulli(exp(0.126,Attacker.Time))
4	Yes	Yes	No	No	bernoulli(exp(0.167,Attacker.Time))
5	Yes	No	Yes	Yes	bernoulli(exp(0.120,Attacker.Time))
6	Yes	No	Yes	No	bernoulli(exp(0.192,Attacker.Time))
7	Yes	No	No	Yes	bernoulli(exp(0.175,Attacker.Time))
8	Yes	No	No	No	bernoulli(exp(0.229,Attacker.Time))
9	No	Yes	Yes	Yes	bernoulli(exp(0.066,Attacker.Time))
10	No	Yes	Yes	No	bernoulli(exp(0.112,Attacker.Time))
11	No	Yes	No	Yes	bernoulli(exp(0.146,Attacker.Time))
12	No	Yes	No	No	bernoulli(exp(0.192,Attacker.Time))
13	No	No	Yes	Yes	bernoulli(exp(0.133,Attacker.Time))
14	No	No	Yes	No	bernoulli(exp(0.192,Attacker.Time))
15	No	No	No	Yes	bernoulli(exp(0.167,Attacker.Time))
16	No	No	No	No	bernoulli(exp(0.263,Attacker.Time))

Table 24: Data on `ExploitSQLInjectionVulnerability`.

Scenario	OPERATOR	BBT	EXPERIENCE	EFFORT	Data
1	Yes	Yes	Yes	Yes	bernoulli(exp(0.058,Attacker.Time))
2	Yes	Yes	Yes	No	bernoulli(exp(0.126,Attacker.Time))
3	Yes	Yes	No	Yes	bernoulli(exp(0.126,Attacker.Time))
4	Yes	Yes	No	No	bernoulli(exp(0.167,Attacker.Time))
5	Yes	No	Yes	Yes	bernoulli(exp(0.120,Attacker.Time))
6	Yes	No	Yes	No	bernoulli(exp(0.192,Attacker.Time))
7	Yes	No	No	Yes	bernoulli(exp(0.175,Attacker.Time))
8	Yes	No	No	No	bernoulli(exp(0.229,Attacker.Time))
9	No	Yes	Yes	Yes	bernoulli(exp(0.066,Attacker.Time))
10	No	Yes	Yes	No	bernoulli(exp(0.112,Attacker.Time))
11	No	Yes	No	Yes	bernoulli(exp(0.146,Attacker.Time))
12	No	Yes	No	No	bernoulli(exp(0.192,Attacker.Time))
13	No	No	Yes	Yes	bernoulli(exp(0.133,Attacker.Time))
14	No	No	Yes	No	bernoulli(exp(0.192,Attacker.Time))
15	No	No	No	Yes	bernoulli(exp(0.167,Attacker.Time))
16	No	No	No	No	bernoulli(exp(0.263,Attacker.Time))

10.2.5. Find Public Command Injection Vulnerability

This a attack step concerns whether an attacker is able to discover a command injection vulnerability in a public domain. To reach this attack step, there is a need for the attacker to be able to connect to the web server that exposes the WA (i.e., `ApplicationServer.ConnectTo` = TRUE).

If connection is possible, this attack step is FALSE if `HasPublicCommandInjection` is FALSE; else, it is TRUE.

10.2.6. Find Public Cross Site Scripting Vulnerability

This a attack step concerns whether an attacker is able to discover an XSS vulnerability in a public domain. To reach this attack step, there is a need for the attacker to be able to connect to the web server that exposes the WA (i.e., `ApplicationServer.ConnectTo` = TRUE).

If connection is possible, this attack step is FALSE if `HasPublicCrossSiteScripting` is FALSE; else, it is TRUE.

10.2.7. Find Public Remote File Inclusion Vulnerability

This a attack step concerns whether an attacker is able to discover an RFI vulnerability in a public domain. To reach this attack step, there is a need for the attacker to be able to connect to the web server that exposes the WA (i.e., `ApplicationServer.ConnectTo` = TRUE).

If connection is possible, this attack step is FALSE if `HasPublicRemoteFileInclusion` is FALSE; else, it is TRUE.

10.2.8. Find Public SQL Injection Vulnerability

This a attack step concerns whether an attacker is able to discover an SQLi vulnerability in a public domain. To reach this attack step, there is a need for the attacker to be able to connect to the web server that exposes the WA (i.e., `ApplicationServer.ConnectTo` = TRUE).

If connection is possible, this attack step is FALSE if `HasPublicSQLInjection` is FALSE; else, it is TRUE.

10.2.9. Discover Vulnerability

This attack step concerns whether an attacker is able to manually discover a novel command injection, XSS, RFI or SQLi vulnerability in a WA. To reach this attack step, there is a need for the attacker to be able to connect to the web server that exposes the WA (i.e., `ApplicationServer.ConnectTo` = TRUE); else, it is FALSE.

If connection is possible, then the likelihood of success depends on four defenses: `TypeSafeAPI` (API), `DeveloperSecurityTraining` (DST), `BlackBoxTesting` (BBT), and `StaticCodeAnalysis` (SCA). How these defenses affect the likelihood of success is shown in Table 25. These data come from [55].

Table 25: Data on `DiscoverVulnerability`.

Project	API	DST	BBT	SCA	Data
1	Yes	Yes	Yes	Yes	bernoulli(linear([0,1.25,4.125,6.625,6.973684], [0,0.05,0.5,0.95,1], Attacker.Time))
2	Yes	Yes	Yes	No	bernoulli(linear([0,2.125,3.625,5.75,6.052632], [0,0.05,0.5,0.95,1], Attacker.Time))
3	Yes	Yes	No	Yes	bernoulli(linear([0,1.875,3.125,5.25,5.526316], [0,0.05,0.5,0.95,1], Attacker.Time))
4	Yes	Yes	No	No	bernoulli(linear([0,0.5,1,1.875,1.973684], [0,0.05,0.5,0.95,1], Attacker.Time))
5	Yes	No	Yes	Yes	bernoulli(linear([0,0.625,1.375,2.25,2.368421], [0,0.05,0.5,0.95,1], Attacker.Time))
6	Yes	No	Yes	No	bernoulli(linear([0,0.625,1.375,2.125,2.236842], [0,0.05,0.5,0.95,1], Attacker.Time))
7	Yes	No	No	Yes	bernoulli(linear([0,0.75,1.5,2.125,2.236842], [0,0.05,0.5,0.95,1], Attacker.Time))
8	Yes	No	No	No	bernoulli(linear([0,0.625,1.125,1.5,1.578947], [0,0.05,0.5,0.95,1], Attacker.Time))
9	No	Yes	Yes	Yes	bernoulli(linear([0,2.125,3.875,6.375,6.710526], [0,0.05,0.5,0.95,1], Attacker.Time))
10	No	Yes	Yes	No	bernoulli(linear([0,1.5,2.5,4,4.210526], [0,0.05,0.5,0.95,1], Attacker.Time))
11	No	Yes	No	Yes	bernoulli(linear([0,1.25,2.25,4.125,4.342105], [0,0.05,0.5,0.95,1], Attacker.Time))
12	No	Yes	No	No	bernoulli(linear([0,0.125,0.375,1,1.052632], [0,0.05,0.5,0.95,1], Attacker.Time))
13	No	No	Yes	Yes	bernoulli(linear([0,0.375,1,2.25,2.368421], [0,0.05,0.5,0.95,1], Attacker.Time))
14	No	No	Yes	No	bernoulli(linear([0,0.125,0.875,1.5,1.578947], [0,0.05,0.5,0.95,1], Attacker.Time))
15	No	No	No	Yes	bernoulli(linear([0,0.25,1,1.5,1.578947], [0,0.05,0.5,0.95,1], Attacker.Time))
16	No	No	No	No	bernoulli(linear([0,0.25,0.625,0.875,0.921053], [0,0.05,0.5,0.95,1], Attacker.Time))

11. Web Application Firewall

While considerable effort has been spent to understand and solve Web Application (WA) security problems many application developers are still unable to implement effective countermeasures for WA vulnerabilities (Scholte et al., 2012).

Consequently, it is imperative for enterprises to effectively manage vulnerabilities in deployed WAs. For this purpose, enterprises often turn to defenses that mitigate vulnerabilities without requiring changes of the application source code. A popular such tool is the `WebApplicationFirewall` (WAF), which has the purpose to prevent attacks on WAs. In CySeMoL, a WAF is a combination of hardware and software (e.g., an OS) that enables WAF functionality. In other words, all concepts related to a WAF are modeled by the asset `WebApplicationFirewall`.

A WAF can be connected to three different assets (see Figure 22). A WAF should be connected to one or more `WebApplications` (denoting that these are protected by the WAF). A WAF can be connected to one or more

AccessControlPoints to denote login functionality of a WAF. Connection to a NetworkZone denotes that the WAF is possible to interface from this NetworkZone (when the two latter are enabled, the WAF can be subjected to being disabled by an attacker).

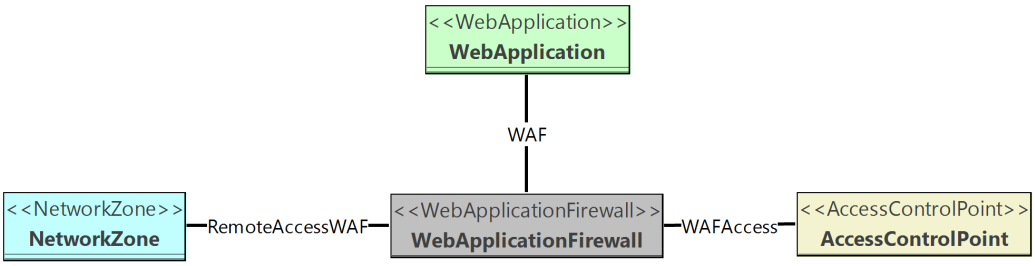

Figure 23: An overview of the connections for WebApplicationFirewall.

There are four defenses corresponding to a WAF; these are shown in Table 26 and desribed in depth in the following sections. The references in Table 26 describe the rationale behind these attributes, e.g., regarding choice of quantitative data.

Table 26: Attack steps and defenses of the WebApplicationFirewall class.

Attribute	Rationale
Defense	
MonitoredByOperator	[56, 51]
TunedUsingBlackBoxTool	[56, 51]
TunedByExperiencedProfessional	[56, 51]
TunedWithSignificantManualEffort	[56, 51]

11.1. Defenses

11.1.1. Monitored By Operator

The presence of an experienced operator monitoring the WAF [56] should make successful WA injection attacks more difficult to perform as many such threats utilize brute-force principles to find vulnerable parameters (e.g., to find if any encoded variants of a single-quote can be passed as an argument to an SQL database). This type of user behavior might not be seen as a threat by a WAF, but will most certainly be viewed with care by an operator.

The default state of this defense is FALSE; this was determined through a case study [51].

11.1.2. Tuned Using Black Box Tool

Employment of an automated black box testing tool for tuning the WAF should serve to decrease both its false positives and false negatives [56]. Such a tool should, for instance, decrease the manual effort required to find all application parameters that can be manipulated by a user of the application.

The default state of this defense is FALSE; this was determined through a case study [51].

11.1.3. Tuned By Experienced Professional

The experience of the individual tuning the WAF should have significant impact on its effectiveness as a more competent individual has a greater understanding of tools, threats and vulnerabilities (and consequently how to mitigate them) [56].

The default state of this defense is TRUE; this was determined through a case study [51].

11.1.4. Tuned With Significant Manual Effort

The effort spent tuning the WAF [56] should be of importance as tuning requires time - not only during the deployment of the WAF, but also for maintenance during the life-cycle of the protected WA. A WAF that receives significant manual effort for tuning should thus be more effective than a WAF that receives little manual effort for tuning.

The default state of this defense is FALSE; this was determined through a case study [51].

12. Data store

A `Datastore` is a place where data can be stored. It could for instance be a relational database, a file server, a processing queue or a database table. A database that has different encryption policies for different variables (e.g., encrypts passwords but not usernames) should be represented as several data stores.

A `Datastore` can be connected to five different assets (cf. Figure 24). A connection to a software (`OperatingSystem`, `ApplicationClient`, `ApplicationServer` or `WebApplication`) denotes that this software is a manager of the data store. A connection to a `Dataflow` denotes that this particular dataflow can interact with the data store (e.g., read or write to it).

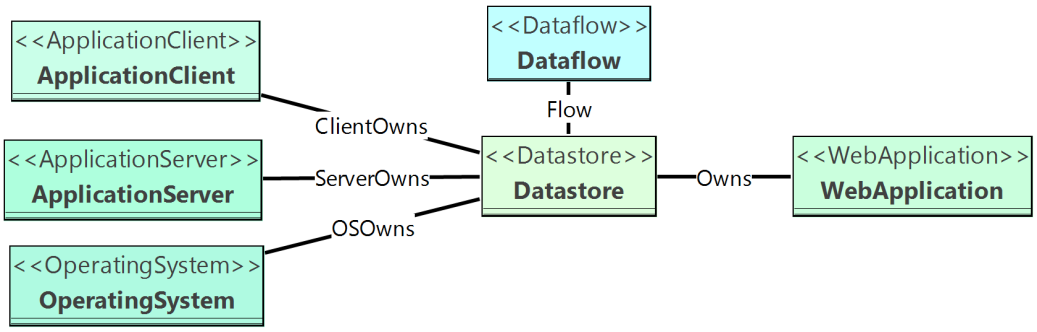

Figure 24: An overview of the connections for `Datastore`.

There are one defense and three attack steps corresponding to data stores; these are shown in Table 27 and desribed in depth in the following sections. The references in Table 27 describe the rationale behind these attributes.

Table 27: Attack steps and defenses of the `Datastore` class.

Attribute	**Rationale**
Defense	
`CryptographicObfuscation`	[57]
Attack step	
`ReadData`	[58, 59]
`WriteData`	[58, 59]
`DeleteData`	[58, 59]

12.1. Defenses

12.1.1. Cryptographic Obfuscation

CySeMoL simplify cryptographic obfuscation of data stores into one variable with two states; either it functions and ensures obfuscation, or it does not function, and ergo does not ensure obfuscation [57]. Attack steps that could make it not function are not detailed, nor are defenses that make such attack steps difficult (e.g. long keys stored securely).

The default state of this defense is TRUE.

12.2. Attack Steps

12.2.1. Read Data

This attack step represents an activity where an attacker can read data in the data store and understand its meaning, i.e. reading encrypted code does not suffice [58, 59].

If a `Dataflow` reads data from a data store and this data flow is eavesdropped (`Dataflow.Eavesdrop` or `Dataflow.ManInTheMiddle`) it also means that the content of the data store is revealed. The content of a data store can also be read by accessing the software that owns it (`OperatingSystem.Access`, `ApplicationClient.Access` or `ApplicationServer.Access`). This could for example be a database management system if the data store is a relational database. Access to this type of software typically means that content in the data store can be read as well. Reading data io a data store can also be accomplished if it is connected to a `WebApplication` that an attacker has compromised with SQL injection (`WebApplication.ExploitSQLInjection`).

If one of these attack steps is TRUE, this attack step is TRUE; else, it is FALSE.

12.2.2. Write Data

If a `Dataflow` updates or changes the data store this data flow can be used to write to it [58, 59].

In CySeMoL, one way for an attacker to accomplish this is to compromise a dataflow that is used to write to the data store. If so, there are two options of attack: either to intercept data in transit and change it before forwarding it to the data store (`Dataflow.ManInTheMiddle`), or simply replaying altered data at some later occassion (`Dataflow.Replay`).

The content of a data store can also be written to by accessing the software that manages it (`OperatingSystem.Access`, `ApplicationClient.Access` or `ApplicationServer.Access`). This could for example be a database management system if the data store is a relational database. Access to this type of software typically means that content in the data store can be written to as well. Writing data to a data store can also be accomplished if it is connected to a `WebApplication` that an attacker has compromised with SQL injection (`WebApplication.ExploitSQLInjection`).

If one of these attack steps is TRUE, this attack step is TRUE; else, it is FALSE.

12.2.3. Delete Data

If a `Dataflow` updates or changes the data store this data flow can be used to delete data in the data store [58, 59].

In CySeMoL, one way for an attacker to accomplish this is to compromise a dataflow that can be used to delete data in the data store. If so, there are two options of attack: either to intercept data in transit and change it before forwarding it to the data store (`Dataflow.ManInTheMiddle`), or simply replaying altered data at some later occassion (`Dataflow.Replay`). The latter could for example involve an attacker that sniffs deletion of data at a time T_1, somebody inserting a new entry at the same place in the database at a time T_2, and the attacker then replaying the traffic to delete this new piece of data at a time T_3.

The content of a data store can also be deleted by accessing the software that manages it (`OperatingSystem.Access`, `ApplicationClient.Access` or `ApplicationServer.Access`). This could for example be a database management system if the data store is a relational database. Access to this type of software typically means that content in the data store can be deleted as well. Deleting data in a data store can also be accomplished if it is connected to a `WebApplication` that an attacker has compromised with SQL injection (`WebApplication.ExploitSQLInjection`).

If one of these attack steps is TRUE, this attack step is TRUE; else, it is FALSE.

13. Data flow

Data, access to data, and how data flows in systems, is the locus in many approaches to security. It is for instance central in the renowned Bell-LaPadula model [60]. A `Dataflow` concerns information that flows beteen two or more applications. In CySeMoL, it does not concern the flow of information within a software application.

There are six assets that can be connected to `Dataflow` in eight different means (cf. Figure 25). A data flow should be connected to a `Protocol` that defines the unique characteristics of the data flow (e.g., in terms of encryption and authentication). This differentiation is made as there often are various data flows of the same type (e.g., HTTP or HTTPS) within organizations. Connection to an `ApplicationClient` denotes that the client is an initiator of data exchange; similarly, connection to an `ApplicationServer` denotes that the server responds to request by clients connected to the data flow. Connection to a `NetworkInterface` denotes that the data flow is allowed to pass through any

Firewall connected to that network interface. There are three possible connections to a NetworkZone: client, server or medium. A server connection implies that a client of the data flow can connect to any remotely accessible application servers within the network zone. In other words, if an ApplicationServer, or an OperatingSystem running this ApplicationServer, is connected to a NetworkZone that has a server connection to a Dataflow, then an attacker with access to an ApplicationClient or NetworkZone connected to this Dataflow (the NetworkZone would require a client connection) can communicate to the network zone or server. Similarly, a client connection implies that an attacker with access to the client NetworkZone (NetworkZone.ObtainAddress = TRUE) can reach any NetworkZones (given a server connection), and ApplicationServers connected to it. Finally, a medium connection implies that a network zone is the medium used by a data flow.

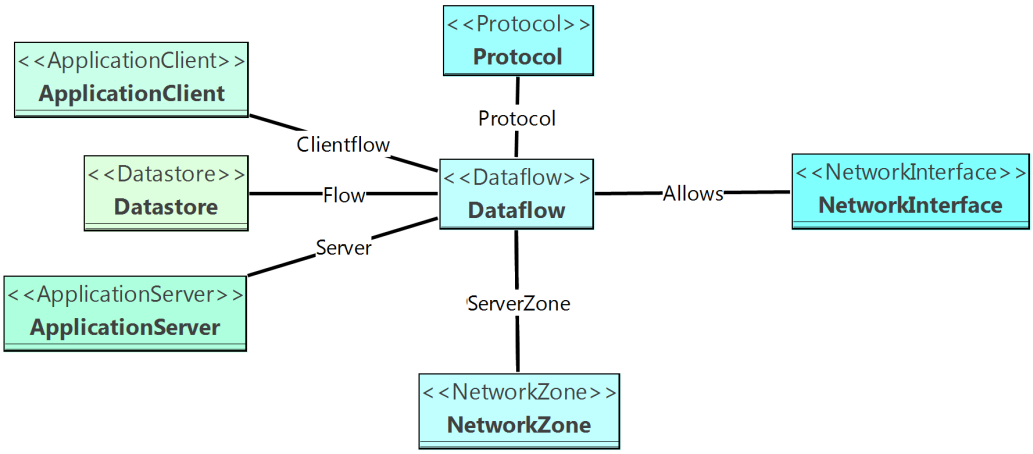

Figure 25: An overview of the connections for Dataflow.

There are six attack steps (and no defenses) corresponding to data stores; these are shown in Table 28 and desribed in depth in the following sections. The references in Table 28 describe the rationale behind these attributes.

13.1. Attack Steps

13.1.1. Disrupt

This attack step represent the possibility that an attacker can disrupt the data flow [24]. This means degrading it so that it influences the functionality it is intended to provide. In CySeMoL, this can be accomplished by either: 1) causing denial of service (DoS) of an application server that is connected to the data

Table 28: Attack steps and defenses of the `Dataflow` class.

Attribute	Rationale
Attack Step	
`Disrupt`	[24]
`Replay`	[61]
`Eavesdrop`	[62]
`ManInTheMiddle`	[63]
`ProduceRequest`	[63]
`ProduceResponse`	[63]

flow (`ApplicationServer.DenialOfService`), 2) causing denial of service for a network interface that allows it (`NetworkInterface.DenialOfService`), 3) ARP-spoofing the server that runs the server with an active data flow.

For instance, SCADA Human Machine Interfaces (HMI) typically pull process data in a dynamic faschion from some historical database. If the purpose is to protect the availability of the dataflow between the historian and the HMI, this attack step is important to consider.

If any of these are TRUE, this attack step is TRUE; else, it is FALSE.

13.1.2. Replay

In the replay attack messages sent previously are sent again [61]. It could for instance be used to send outdated process measurements to a control center that fool operators that the power system state is at is was yesterday (when it is not).

In CySeMoL, this attack step requires access to either a client or server connected to the data flow in question (`ApplicationClient.Access` or `ApplicationServer.Access`), or an IP on a network zone that acts as a medium to it (`NetworkZone.ObtainOwnAddress`). In addition, it requires the `Protocol` connected to the data flow to not have freshness indicators functioning (`Protocol.FreshnessIndicator` = FALSE) [61].

Given these circumstances, this attack step is TRUE; else, it is FALSE.

13.1.3. Eavesdrop

Message eavesdropping means that the threat agent can read the content sent in this data flow [62].

In CySeMoL, eavesdropping is successful if the attacker has access to either the client or server that exchanges information (`ApplicationClient.Access`

or ApplicationServer.Access). If not, there is a need for the attacker to have an IP on a network zone that acts as a medium to the data flow (NetworkZone.ObtainOwnAddress), and that the Protocol connected to the data flow does not have cryptographic obfuscation enabled (Protocol.CryptographicObfuscation = FALSE)

Given these circumstances, this attack step is TRUE; else, it is FALSE.

13.1.4. Man In the Middle

In a man in the middle (MITM) attack messages are altered during a communication session. The attacker intercepts messages, alters them and resubmits them [63]. The purpose of the attack is to alter the content of the communication while making the receiver think the message is an original.

One possibility of succeeding with this attack step is if the attacker has compromised one of the actors participating in the communication (ApplicationClient.Access or ApplicationServer.Access) (*AC/AS*); if so, this attack step is TRUE.

If not, MITM can still be accomplished by ARP-spoof of an OS running an actor connected to the targeted data flow (OperatingSystem.ARPSpoof), or DNS-spoofing a network zone that acts as a medium to the data flow (NetworkZone.DNSSpoof) (*ARP/DNS*). However, for these attacks to be successful, there is a need for the Protocol connected to the data flow to not have cryptographic obfuscation (Protocol.CryptographicObfuscation) or cryptographic authentication (Protocol.CryptographicAuthentication) enabled (*CO/CA*)). If ARP-spoof or DNS-spoof are TRUE, and no cryptographic obfuscation or authentication is used, then the likelihood of success is 99%. This estimate was gained from an interview with a domain expert in combination with vulnerability data from the US National Vulnerability Database (NVD).

An overview of this logic can be found in Table 29.

13.1.5. Produce Request

This attack step involves if an attacker can produce messages that correspond to a requests from a client to a data flow. This could be done by establishing a session and send whole messages, or by injecting messages in an on-going session [63].

In CySeMoL, this can be accomplished by a successful MITM attack (ManInTheMiddle), or by simply compromising the client part of a data flow. This can be either obtaining an address on a network zone that is client to a data

Table 29: Defenses and attack steps affecting likelihood of `ManIntheMiddle`.

AC/AS	CO/CA	ARP/DNS	Data
TRUE	TRUE	TRUE	bernoulli(1)
TRUE	TRUE	FALSE	bernoulli(1)
TRUE	FALSE	TRUE	bernoulli(1)
TRUE	FALSE	FALSE	bernoulli(1)
FALSE	TRUE	TRUE	bernoulli(0)
FALSE	TRUE	FALSE	bernoulli(0)
FALSE	FALSE	TRUE	bernoulli(0.99)
FALSE	FALSE	FALSE	bernoulli(0)

flow (`NetworkZone.ObtainAddress`), or by compromising a software client that is connected to the targeted data flow (`ApplicationClient.Access`).

Given these circumstances, this attack step is TRUE; else, it is FALSE.

13.1.6. Produce Response

This attack step involves if an attacker can produce messages that correspond to a responses by a server to requests from a client to a data flow. This could be done by establishing a session and send whole messages, or by injecting messages in an on-going session [63].

In CySeMoL, this can be accomplished by a successful MITM attack (`ManInTheMiddle`), or by simply compromising the server part of a data flow. This can be either obtaining an address on a network zone that is server to a data flow (`NetworkZone.ObtainAddress`), or by compromising a software server that is connected to the targeted data flow (`ApplicationServer.Access`).

Given these circumstances, this attack step is TRUE; else, it is FALSE.

14. Protocol

A `Protocol` contains the security-relevant properties of data flows, and thus what makes different types of data flows unique. Consequently, a data flow should always be depicted along the protocol that defines it. The differentiation between data flow and protocol is made as there often are various data flows of the same type (e.g., HTTP or HTTPS) within organizations. The `Dataflow` is the only asset that can be connected to a protocol (cf. Figure 24).

Figure 26: An overview of the connections for `Protocol`.

There are three defenses corresponding to data flows; these are shown in Table 30 and desribed in depth in the following sections. The references in Table 30 describe the rationale behind these attributes.

Table 30: Attack steps and defenses of the `Protocol` class.

Attribute	**Rationale**
Defense	
`FreshnessIndicator`	[64, 65]
`CryptographicAuthentication`	[64, 65]
`CryptographicObfuscation`	[64, 65]

14.1. Defenses

14.1.1. Freshness Indicator

The purpose of a freshness indicator is to denote the relative order or time that a message was sent, this as messages both can be lost in transit or recieved in an unintended order. Or in the case of a cyber attack - tampered with. Three types of commonly employed freshness indicators include sequence numbers, nonces and timestamps [64, 65].

The default state of this defense is TRUE as most protocols (e.g., TCP/IP) support some notion of it.

14.1.2. Cryptographic Authentication

CySeMoL simplifies cryptographic authentication embedded in protocols into one variable with two states. Either it functions and ensures authentication, or it does not function. Attack steps that could make it not function are not detailed, nor are defenses that makes such attack steps difficult (e.g., long keys). Broken cryptographic protocols, such as Wired Equivalent Privacy (WEP), should be modeled as `CryptographicAuthentication` = FALSE as they do not ensure authentication.

The default state of this defense is FALSE as many protocols do not require authentication per default.

14.1.3. Cryptographic Obfuscation

CySeMoL simplifies cryptographic obfuscation embedded in protocols into one variable with two states. Either it functions and ensures obufuscation, or it does not function. Attack steps that could make it not function are not detailed, nor are countermeasures that makes such attack steps difficult (e.g. long keys). Broken cryptographic protocols, such as WEP, should be modeled as `CryptographicObfuscation` = FALSE as they do not ensure obfuscation.

The default state of this defense is FALSE as many protocols do not require authentication per default.

15. Network Zone

A `NetworkZone` enables a logical means for a set of hosts to reach each other. This could for example be a local area network (LAN) or a wide area network (WAN).

A `NetworkZone` can be connected to twelve different assets in 17 different means (cf. Figure 27). A connection to a software (`OperatingSystem`, `ApplicationClient`, or `ApplicationServer` or `WebApplication`) denotes that this software is directly reachable over the network. Connections to `ApplicationServer` is however not necessary as these automatically are reachable if their corresponding `OperatingSystem` is connected to the `NetworkZone`. In other words, it is assumed that an attacker that can remotely interface with core services of an OS also can interface with other services (e.g., HTTP or SMTP servers) running on the OS.

A connection to a `IDS`, `IPS`, `Firewall` or `WebApplicationFirewall` denotes that this device has a logical login function that is remotely reachable from the network zone.

There are three possible connections to a `Dataflow`: client, server or medium. A server connection implies that a client of the data flow can connect to any remotely accessible application servers within the network zone. In other words, if an `ApplicationServer`, or an `OperatingSystem` running this `ApplicationServer`, is connected to a `NetworkZone` that has a server connection to a `Dataflow`, then an attacker with access to an `ApplicationClient` or `NetworkZone` connected to this `Dataflow` (the `NetworkZone` would require a client connection) can communicate to the network zone or server. Similarly, a client connection implies that an attacker with access to the client `NetworkZone` (`NetworkZone.ObtainAddress` = TRUE) can reach any `NetworkZones` (given a

server connection), and ApplicationServers connected to it. Finally, a medium connection implies that a network zone is the medium used by a data flow.

There are two possible connections to NetworkInterface: untrusted zone and trusted zone. An untrusted zone notifies that any information going from this zone through the network interface is not trusted; similarly, a trusted zone means that any information going from this zone through the network interface can be considered trusted. This delimitations is made to remove a direct cycle involving an attacker going from one zone to another. This semantic difference affects various attack steps in CySeMoL. For example, if an IDSSensor is connected to a NetworkInterface, it will inspect traffic going from its corresponding untrusted zones to its trusted zones, but not the other way around.

Network zones often coincide with the notion of network segments. The set of hosts in such a segment/zone often have the same level of security associated them - they are under the same security policy. For example with respect to patching, hardening levels, physical security and user access levels. This is addressed by the asset ZoneManagementProcess; if this is connected to a network zone it will influence the states of various defenses within the zone.

Similarly to ZoneManagementProcess, a network zone can be connected to a NetworkVulnerabilityScanner. This connection signifies that the zone regularly undergoes either authenticated or unauthenticated scans by a network scanner in order to find and eventually mitigate vulnerabilities in devices present on the network zone.

Finally, connection to a PhysicalZone denotes that an attacker has direct physical access to the network zone.

There are two defenses and four attack steps corresponding to network zones; these are shown in Table 31 and desribed in depth in the following sections. The references in Table 31 describe the rationale behind these attributes.

15.1. Defenses

15.1.1. DNS Sec

The domain name system is a hierarchical system involving a large number of domain name servers organized in a tree-architecture. These servers map URLs to IP-addresses. Nodes are divided into DNS zones which are collectively served by a domain name server. Domain name servers may also delegate parts of its DNS zone to other domain name servers.

So, a network zone's hosts will belong to one or more DNS zones and they will query the domain name servers for IP addresses corresponding to different URLs. To resolve such queries the domain name server might query other domain

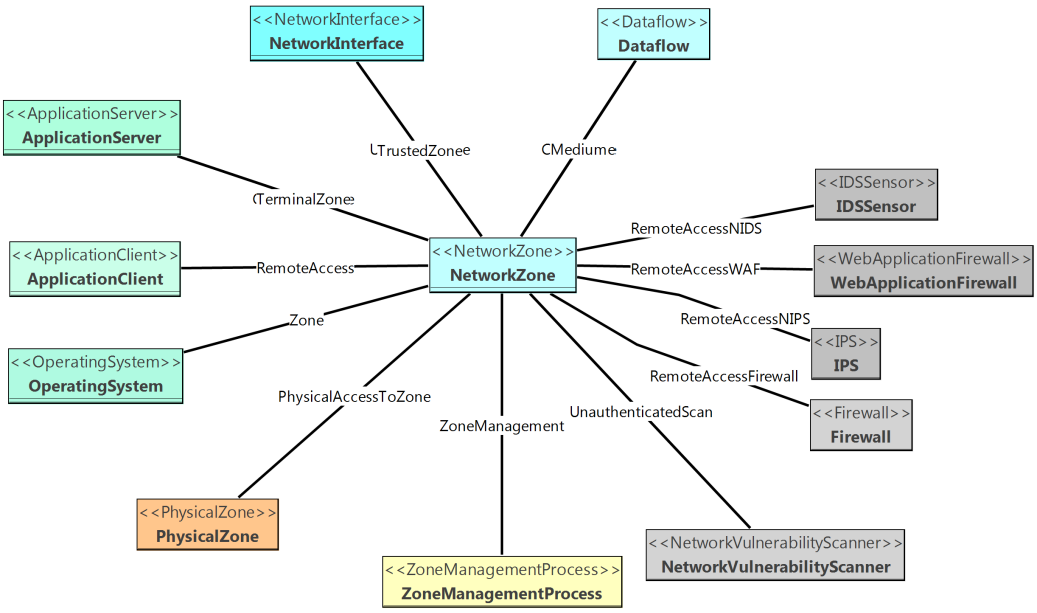

Figure 27: An overview of the connections for `NetworkZone`.

name servers for the requested information. DNS servers thus communicate with each other and with the clients requesting domain names to be resolved. The Domain Name System Security Extensions (DNSSEC) adds authentication to this communication through digital signatures [66]. The purpose is to protect against DNS spoofing attacks where the DNS cache is manipulated in a server.

The default state of this defense is TRUE as DNSsec is common in practice.

15.1.2. Port Security

Port security (also known as Port Binding or MAC Binding) is related to the network devices used in the network zone, such as switches. Port security means that the ports (network outlets) on a network device are locked to a set of specified MAC (Media Access Control) addresses [67].

While a computer's MAC address can be changed to one of the permitted addresses this offer some protection against attacks where the threat agent physically connect to the network. If port security is activated the threat agent must identify a permitted MAC address and change the address of the compute one wish to connect. To find the right address is not trivial without network access. Also, the right MAC address owner must be disconnected and replaced to enable use of their MAC address. If there is a one-to-one mapping between outlets and

Table 31: Attack steps and defenses of the NetworkZone class.

Attribute	Rationale
Defense	
DNSSec	[66]
PortSecurity	[67]
Attack step	
DNSSpoof	[66]
DenialOfService	[23, 24]
FindUnknownEntryPoint	[27, 25]
ObtainOwnAddress	[66]

MAC addresses this is definitely necessary. But even if there is no such one-to-one mapping a conflict will arise if two identical MAC addresses are within the same network.

The mapping between MAC addresses and switches' outlets can be a one-to-one mapping, or it can specify more general requirements; for instance, what MAC addresses that can be connected to a specific switch or a set of switches.

The default state of this defense is FALSE.

15.2. Attack Steps

15.2.1. DNS Spoof

DNS spoofing means that an attacker associates a URL with a desired IP address. In [66] three types of DNS spoofing attacks are discussed:

- Cache poisoning - a server is served with manipulated IP to URL-mappings that are stored in its cache.
- Server compromising - the DNS server is compromised by the attacker.
- Spoofing - the attacker masquerades its own DNS server as some other (legitimate) DNS server

All these are included in this attack step. To succeed with the attack, there is a need for the attacker to reach the network zone as such (ObtainAddress = TRUE). A condition making cache poisoning more difficult is if DNSSec is used in the zone. If DNSSec is used, the likelihood of this attack step is FALSE. If the attacker can reach the network zone and DNS sec is false, this attack step is TRUE.

15.2.2. Denial Of Service

This attack step concerns causing denial of service for the network zone communication medium [23, 24].

In CySeMoL, this can be accomplished if an attacker can ARP spoof the network interface connected to the network zone (`NetworkInterface.ARPSpoof`). If not, it can still be accomplished if the attacker is able to reach the network zone as such (`ObtainAddress`). If so, the likelihood of successful attack is 0.1%. This was estimated based on an interview with a domain expert [24]. If the attacker cannot reach the network zone, this attack step is FALSE.

15.2.3. Find Unknown Entry Point

It is difficult to secure a system which is largely or partly unknown. To build and maintain an (cyber) asset inventory is therefore often recommended [68].

This attack step groups actions that identify entry points to a network zone which are not known to the persons responsible creating the instance model. To accomplish this attack step one could for instance find and use:

- Data flows allowed by the firewall, while the model creator think they are not.
- Connections that are not known to exist, e.g. modems that are thought to be disconnected.
- Find a host with dual network interfaces that were believed to belong in only one network zone.

The probability that this attack step can be accomplished depends on the attacker's ability to find existing entry points. But it also depends on the gap between the instance model and the real world. The gap between the instance model and the real system can be reduced by investing effort in keeping the model (and the security group's knowledge) up-to-date. The ideal situation is when the change management process is functioning and the model is kept updated.

To accomplish this attack step, there is a need for an attacker to be able to probe the zone for possible entry points. In CySeMoL, this means that the attacker must have an address on an adjacent `NetworkZone` (a network zone connected to the same `NetworkInterface` as the probed zone).

If this is the case, the likelihood of gaining an entry depends on whether there is a `Firewall` or not, and whether this firewall has been compromised, is functioning (`Firewall.Functioning`) or has a modeled rule set that is known to

perfectly reflect the real world (`Firewall.KnownRuleSet`). If the firewall is not functioning, or if it compromised, ths attack step will be TRUE. If the firewall has a known rule set, it will be FALSE. If the firewall is functioning, but it is unknown whether its rule set reflects the real world, the likelihood of attack success depends on how the network zone is maintained.

In CySeMoL, three defenses related to network maintenance influence the likelihood of this attack step being true:

1. If security audits are performed on a regular basis.
2. If logs are reviewed on a regular basis.
3. If there is a formal process for managing changes in the IT architecture.

These correspond to the CySeMoL defenses:

1. `ZoneManagementProcess.RegularSecurityAudits` (Audits)
2. `ZoneManagementProcess.RegularLogReviews` (Logs)
3. `ZoneManagementProcess.FormalChangeControlProcess` (Changes)

How these defenses influence the likelihood of this attack step being TRUE are described in Table 32. These data come from [27, 25].

Table 32: Defenses affecting likelihood of `FindUnknownEntryPoint`.

Audits	**Logs**	**Changes**	**Data**
TRUE	TRUE	TRUE	bernoulli(0.64)
TRUE	FALSE	TRUE	bernoulli(0.65)
FALSE	TRUE	TRUE	bernoulli(0.59)
FALSE	FALSE	TRUE	bernoulli(0.61)
TRUE	TRUE	FALSE	bernoulli(0.65)
TRUE	FALSE	FALSE	bernoulli(0.63)
FALSE	TRUE	FALSE	bernoulli(0.68)
FALSE	FALSE	FALSE	bernoulli(0.71)

15.2.4. Obtain Own Address

Having an address on a network zone means that the attacker can communicate to any device connected to this network zone [66]. This attack step can be accomplished by either 1) compromising an OS on the network zone

(OperatingSystem.Access), 2) compromising a data flow that devices on a network zone is server to (Dataflow.ProduceRequest), 3) finding an unknown entry point to the network zone (FindUnknownEntryPoint).

If any of these attack steps are TRUE, this attack step is TRUE; else, it is FALSE.

16. Network Interface

A NetworkInterface can be seen as the network gateway. On its own, it provides no firewall functionality what so ever - it merely serves to redirect any network traffic to its designated addresses.

A NetworkInterface can be connected to five different assets in six different means(cf. Figure 28). A connection to IDS, IPS or Firewall denotes that traffic through this network interface is managed/analyzed by that device. Connection to a Dataflow denotes that the data flow is allowed to pass through any Firewall connected to the network interface.

There are two possible connections to NetworkZone: untrusted zone and trusted zone. An untrusted zone notifies that any information going from this zone through the network interface is not trusted; similarly, a trusted zone means that any information going from this zone through the network interface can be considered trusted. This delimitations is made partly to remove a direct cycle involving an attacker going from one zone to another, and partly as some information flows are inheretly more trustworthy than others. For instance, the information going from a business network to a supervisory control and data acquisition (SCADA) network is typically less trustworthy than information going from a SCADA network to a business network. This semantic difference affects various attack steps in CySeMoL. For example, if an IDSSensor is connected to a NetworkInterface, it will inspect traffic going from its corresponding untrusted zones to its trusted zones, but not the other way around.

There are one defense and two attack steps corresponding to network interfaces; these are shown in Table 27 and desribed in depth in the following sections. The references in Table 33 describe the rationale behind these attributes.

16.1. Defenses

16.1.1. Static ARP Tables

CySeMoL assumes that the TCP/IP-stack is used and that hosts are identified with MAC-addresses on the physical layer. Address resolution protocol (ARP)

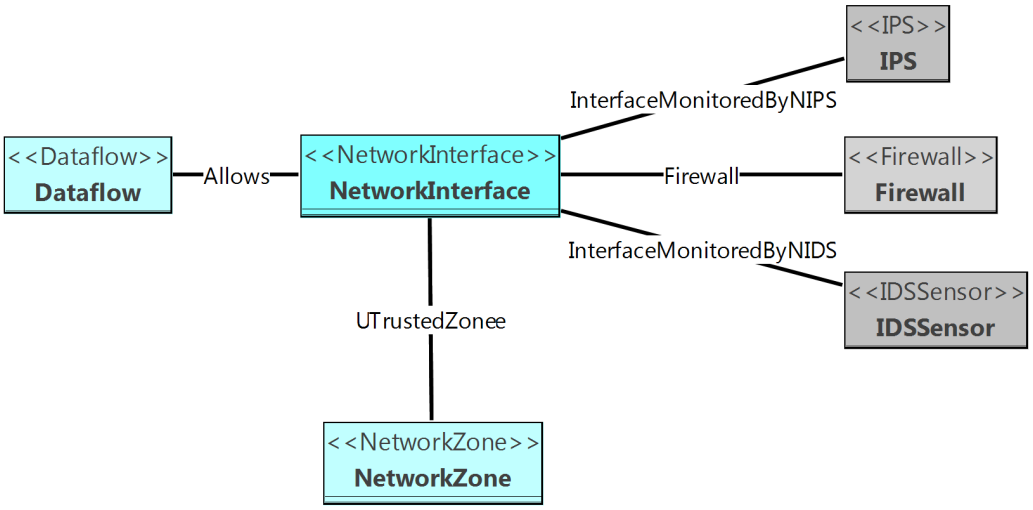

Figure 28: An overview of the connections for `NetworkInterface`.

Table 33: Attack steps and defenses of the `NetworkInterface` class.

Attribute	Rationale
Defense	
`StaticARPTables`	[16]
Attack step	
`ARPSpoof`	[16]
`DenialOfService`	[23, 24]

map MAC-addresses to IP-addresses within the zone. If this mapping is not static it can be compromised by other in the same broadcast domain [16].

The default state of this defense is FALSE as static ARP tables due to number of reasons are impractical [16], and thus not often used.

16.2. Attack Steps

16.2.1. ARP Spoof

This attack step concerns whether it is possible to poison ARP tables in the network interface. If the network interface's ARP tables are poisoned this can be used to intercept traffic going from the external zone to the internal zone [16].

The network interface will use its ARP tables to identify what MAC address an incoming IP package should be forwarded to. If an attacker can compromise the

ARP table then this attacker can make these IP packages end up at arbitrary MAC address (e.g. his/her own). The attacker could then alter data before forwarding it to its intended address.

For this attack step to be successful, there is a need for the attacker to have an address on a network zone trusted by the network interface ((trusted) `NetworkZone.ObtainAddress`). There is also a need for `StaticARPTables` to be FALSE. Given these circumstances, this attack step is TRUE; else, it is FALSE.

16.2.2. Denial Of Service

This attack step concerns causing denial of service for the network interface (e.g., turn it off or overload its message cue and cause major packet loss) [23, 24].

In CySeMoL, this can be accomplished if an attacker can ARP spoof the network interface (`ARPSpoof`). If not, it can still be accomplished if the attacker is able to reach the network zone as such (`NetworkZone.ObtainAddress`). If so, the likelihood of successful attack is 0.1%. This was estimated based on an interview with a domain expert [24]. If the attacker cannot reach the network zone, this attack step is FALSE.

17. Firewall

In general, the meaning and functionality of a firewall can vary significantly. For instance, in [69] firewalls are divided into three classes: screening routers, application proxies and stateful inspectors. These three basic functionalities are also discussed in [70], in addition to a number of other firewall functionalities, e.g., NAT routers, application based firewalls and host based firewalls. Some of these are combinations of functionalities, and some classes relate to where the firewall is deployed (i.e. network interface/gateway or host). In CySeMoL, host firewalls are included in the `OperatingSystem` class; this class concerns *network based firewalls*.

More specifically, a `Firewall` represents a type of packet filter that performs statful inspection and packet analysis. A packet filter (a.k.a. a screening router) focus on inspecting the packet headers origin (IP and port), destination (IP and port) and the transport protocol used [70]. A stateful inspection firewall is functionality added on top on packet filtering. It also keeps track of the state of sessions and that packages session flags conform to these. This check will improve on the ability to detect injected (spoofed) packages [70]. As almost all firewalls used today have both functionalities they are here combined.

A `Firewall` can be connected to three different assets (cf. Figure 29). A connection to a `NetworkZone` denotes that the firewall has a logical login function that is remotely reachable from the network zone. Connection to a `NetworkInterface` denotes that traffic through this network interface is handled by the firewall. Connection to an `AccessControlPoint` described the logical login function of the firewall.

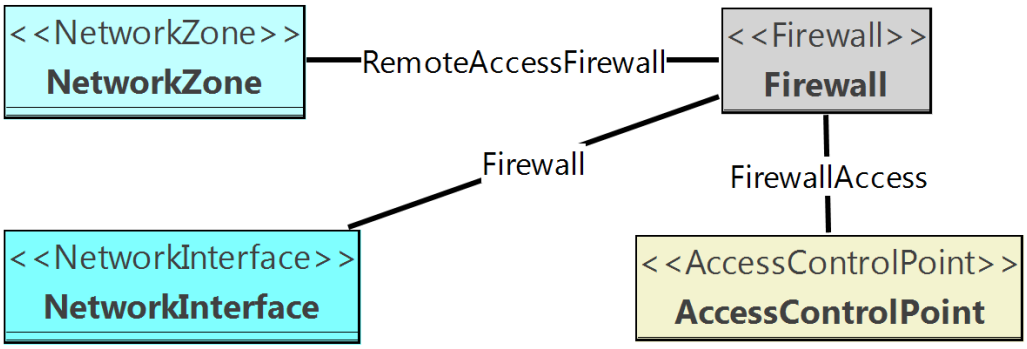

Figure 29: An overview of the connections for `Firewall`.

There are two defenses corresponding to firewalls; these are shown in Table 34 and desribed in depth in the following sections. The references in Table 34 describe the rationale behind these attributes.

Table 34: Attack steps and defenses of the `Firewall` class.

Attribute	**Rationale**
Defense	
`Functioning`	[70]
`KnownRuleSet`	[27, 25]

17.1. Defenses

17.1.1. Functioning

This defense conserns whether the firewall is functioning at all, i.e., whether it performs stateful inspection and packet filtering or not [70].

The default state of this defense is TRUE.

17.1.2. Known Rule Set

This defense concerns whether the modeled rule set of the firewall correctly reflects the real world or not. That is, a firewall might be misconfigured without the knowledge of the individual who creates the CySeMoL instance model [27, 25].

The default state of this defense is FALSE as it is a difficult and time-demanding task to ascertain that a model reflects reality. It should only be set as TRUE in circumstances where there is a very high degree of certainty regarding the firewall's maintenance and function.

18. Intrusion Prevention System

An intrusion prevention system (`IPS`) is a device (a combination of software and hardware) that inspects the application layer and performs stateful protocol analysis [69]. The main difference between a `Firewall` and an `IPS` is that the IPS performs stateful inspection based on not only header- , but also packet payload information.

This firewall functionality of an IPS can, by inspecting the application layer, remove suspicions content, ban certain types of commands, or ban certain combinations of commands. They can for example remove executable files from email attachments or prevent the put-function in FTP (File Transfer Protocol) from being used [69]. Application firewalls can also be used as application proxies to services. They can also identify and ban traffic that is sent repeatedly over the network interface [69].

An `IPS` can be connected to six different assets (cf. Figure 30). A connection to a `NetworkZone` denotes that the IPS has a logical login function that is remotely reachable from the network zone. Connection to a `NetworkInterface` denotes that traffic through this network interface is analzyed by the IPS. Connection to an `AccessControlPoint` described the logical login function of the IPS. Connection to an `ApplicationClient`, `ApplicationServer` or `OperatingSystem` denotes that the IPS is an application proxy to this software (or if OS, collection of software).

There is one defense corresponding to IPSs; this is shown in Table 35 and described in the following section. The reference in Table 35 describe the rationale behind this defense.

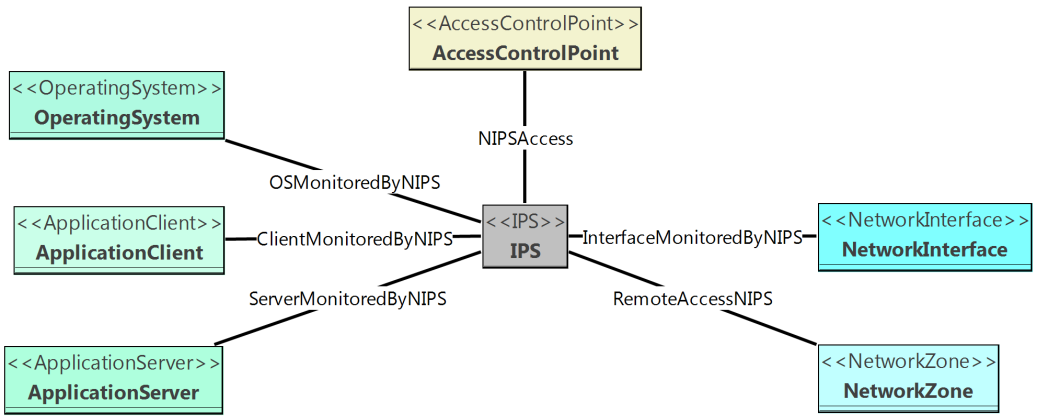

Figure 30: An overview of the connections for `IPS`.

Table 35: Attack steps and defenses of the `IPS` class.

Attribute	Rationale
Defense	
`Functioning`	[70]

18.1. Defenses

18.1.1. Functioning

This defense conserns whether the IPS is functioning at all, i.e., whether it performs stateful inspection and packet filtering or not [70].

The default state of this defense is TRUE.

19. Intrusion Detection System

The plethora of intrusion detection methods and techniques that have been introduced are commonly categorized as either anomaly based or signature (a.k.a. misuse) based [71]. Anomaly based schemes estimates the normal behavior of a system and generates alarms when the deviation from the normal exceeds some threshold [71]. Signature based schemes look for patterns (signatures) in the analyzed data and raise alarms if the patterns match known attacks [71]. CySeMoL includes signature-based IDS as these typically are used in practice [72].

Two common intrusion detection systems (IDS) include network intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS) [71]. HIDS supervise the systems they are deployed on; NIDS supervise the network

traffic on they employed on the perimeter dividing two or more zones. In CySeMoL, the connections of an `IDSSensor` depict whether a HIDS or a NIDS is concerned.

An `IDSSensor` can be connected to four assets (cf. Figure 31). A connection to a `NetworkZone` denotes that the IDS has a logical login function that is remotely reachable from the network zone. Connection to a `NetworkInterface` denotes that traffic through this network interface is analyzed by the IDS (i.e., the IDS is a NIDS). Connection to an `OperatingSystem` denotes that the IDS is a host-based solution on this OS (i.e., a HIDS). Connection to an `AccessControlPoint` described the logical login function of the IDS.

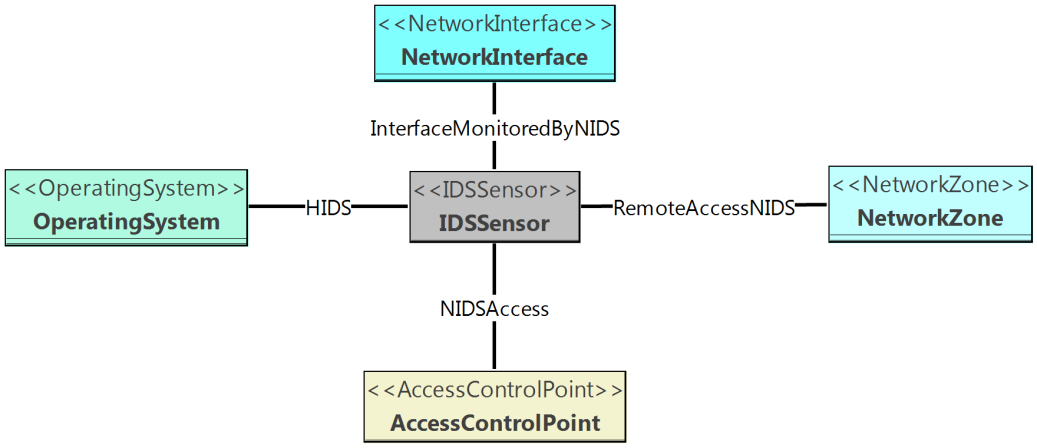

Figure 31: An overview of the connections for `IDSSensor`.

There are two defenses corresponding to IDSs; these are shown in Table 36 and desribed in depth in the following sections. The references in Table 36 describe the rationale behind these attributes.

Table 36: Attack steps and defenses of the `IDSSensor` class.

Attribute	Rationale
Defense	
`Functioning`	[56]
`Tuned`	[56]
`Updated`	[56]

19.1. Defenses

19.1.1. Functioning

This defense conserns whether the `IDSSensor` is properly installed and configured, i.e. that it does what could be expected from an IDS [56].

The default state of this defense is TRUE.

19.1.2. Tuned

Intrusion detection systems are often tuned in to their environment [56]. This could for example include giving the IDS information about the network it is placed on and its normal characteristics.

The default state of this defense is TRUE.

19.1.3. Updated

Signature based IDSs use a ruleset to identify attacks. To be able to detect new attacks the ruleset needs to be updated with signatures that cover those attacks [56]. As with software patches it is often possible to select the updates to apply. However, common practice is to update the ruleset completely when it is updated. This attribute states if the ruleset is completely updated. That is, if the rules used by the signature based IDS are the latest rules that can be applied.

The default state of this defense is TRUE. However, it greatly depends on the context of the scenario. Given a scenario where rule set updates require local access of the IDS (e.g., in many critical infrastructure environments) these are likely also less frequent.

20. Network Vulnerability Scanner

A `NetworkVulnerabilityScanner` is a commonly used tool to identify vulnerabilities such as unpatched software and weak passwords [73, 15].

A `NetworkVulnerabilityScanner` can be connected to a two different assets in five different ways (cf. Figure 32). A connection to a `NetworkZone` or `OperatingSystem` can be enabled denoting either an authenticated or unauthenticated scan. During an unauthenticated scan, the scanner probes for vulnerabilities that are testable through TCP or UDP without any privileges on the studied systems - i.e., any `ApplicationServer` (e.g., an FTP service) connected to the probed `OperatingSystem`. If an `ApplicationServer` has a login interface (e.g., SSH or FTP), the scanner can also attempt to evaluate any poor passwords for this interface. During an authenticated scan, the scanner is allowed to log in to the probed systems. Thus, an authenticated scan

is typically both more effective and can not only evaluate vulnerabilities for the `ApplicationServer`, but also for any `ApplicationClient` (e.g., a web browser) residing on the probed `OperatingSystem`. Both scanning types can also help find `ApplicationServers` unknown to the network administrator (`OperatingSystem.FindUnknownService`). An `OperatingSystem` can also be designated to *not* be part of the scanning policy (this is common in practice as scans can cause availability issues).

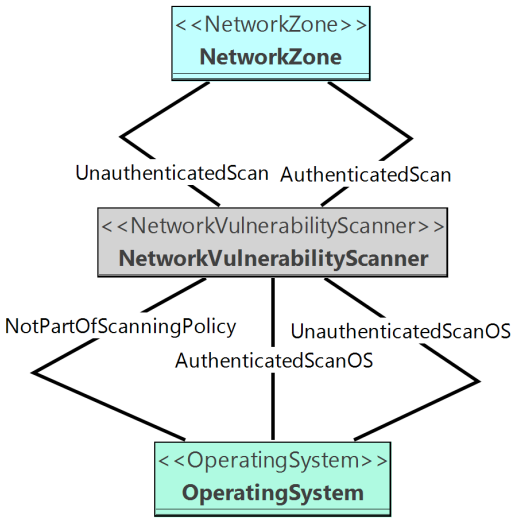

Figure 32: An overview of the connections for `NetworkVulnerabilityScanner`.

There is a single defense corresponding to vulnerability scanners; this is shown in Table 37 and desribed in depth in the following section. The reference in Table 37 describe the rationale behind this attribute.

Table 37: Attack steps and defenses of the `NetworkVulnerabilityScanner` class.

Attribute	Rationale
Defense	
`Functioning`	[73, 15]

20.1. Defenses

20.1.1. Functioning

This defense conserns whether the vulnerability scanner is functioning at all, i.e., whether it is used or not [73, 15].

The default state of this defense is TRUE.

21. Zone Management Process

A `ZoneManagementProcess` represents a process for managing a network zone. One such process might be used to manage multiple network zones. They are here assumed to cover all hosts and installed software in the zone. However, one could imagine cases were several different management processes are used for managing different parts of a network zone. To manage this type of scenario using CySeMoL, there is a need to divide such a network zone into several sub-zones. Numerous of attributes can be associated with IT management activities. COBIT [165], for example, describes 34 processes for controlling IT, over 100 pieces of information that should be passed between these processes and the roles of 15 organizational functions in the processes. CySeMoL simplify this domain into one process and a handful of variables of particular importance.

There is a single connection available for `ZoneManagementProcess` (cf. Figure 33). This connection denotes that this particular `NetworkZone` is managed by the management processes specified in that particular `ZoneManagementProcess`.

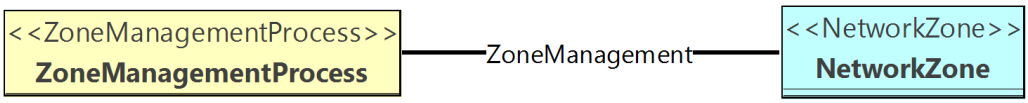

Figure 33: An overview of the connections for `ZoneManagementProcess`.

There are seven defenses corresponding to management processes; these are shown in Table 38 and desribed in depth in the following sections. The references in Table 38 describe the rationale behind these attributes.

21.1. Defenses

21.1.1. Host Hardening Procedures

Procedures for hardening hosts are an often recommended [74, 75, 70, 27]. This activity involves disabling software ports, disabling unused or dangerous services and disabling outlets for portable media. The existence of this defense states that such practices are included in the network management process.

The default state of this defense is FALSE as it is a costly and presumably not very common activity. However, it greatly depends on the actual environment. Given an environment where the functionality of hosts is well known (e.g. control centres), this practice is easier to implement than in environments with more variation and more hosts (e.g. office networks).

Table 38: Attack steps and defenses of the ZoneManagementProcess class.

Attribute	Rationale
Defense	
HostHardeningProcedures	[74, 75, 70, 27]
AutomatedPatchManagementProcess	[14]
RegularLogReviews	[76, 27]
RegularSecurityAudits	[77, 27]
FormalChangeManagementProcess	[22, 75, 27]
ManagedByAntiMalwareSolution	[20]
USBAutoRunDisabledInDomain	[21, 38, 39]

21.1.2. Automated Patch Management Process

When a software vulnerability has been found the software's vendor often develop and distribute an update or patch to remove the vulnerability. This defense involves whether the process of applying such a software update is automated or not [14]. Given an automated patch management process, users are not required to actively download and install critical security updates - this is automatically accomplished by the software developer or an in-house service. The Windows update is an example of this type of mechanism.

The default state of this defense is TRUE as most common commercial-of-the-shelf software are prospect to automatic updates. However, it naturally depends on the context of the enviroment: given a scenario where no internet connection is available (e.g., in control centres), automatic updates is naturally problematic to achieve.

21.1.3. Regular Log Reviews

Regular log reviews is an often recommended practice. One of the primary purposes of audit logs is to validate that the system operates according to policies [76, 27]. Audit logs can for example be used to identify misconfigured firewalls or to resolve problems with security policy compliance. The frequency of conducted reviews is naturally of importance. In CySeMoL, "regularity" refers to that logs are reviewed at least once every 90 calendar days (this is the same recommendation as is given by NERC CIP [74]).

The default state of this defense is TRUE.

21.1.4. Regular Security Audits

`RegularSecurityAudits` involves if the network zone undergoes regular security audits by professionals, an often recommended practice to identify both technical and managerial vulnerabilities [77, 27].

The default state of this defense is FALSE.

21.1.5. Formal Change Management Process

`FormalChangeManagementProcess` involves whether there is a formal process for managing changes in the configuration of an architecture; the greater the understanding of the systems that need be secured, the greater the possibility to identify and mitigate vulnerabilities corresponding to them [22, 75, 27].

The default state of this defense is TRUE. However, as for the other processes within `ZoneManagementProcess`, its state naturally depends on the context of the environment.

21.1.6. Managed By Anti Malware Solution

Anti malware, or anti-virus, solutions is an often recommended practice to mitigate malware [20]. This defense concerns whether there is a domain policy regarding usage of anti-virus solutions within a network zone.

The default state of this defense is TRUE as most enterprises have domain policies regarding anti-virus solutions. However, it should be set to FALSE given an environment without such a policy (e.g., many control center environments).

21.1.7. USB AutoRun Disabled In Domain

Many malware spread through USB drives using the USB AutoRun functionality available in many operating systems [21, 38, 39]. Disabling such functionality thus effectively mitigate this attack vector. This can typically be accomplished on a domain level, e.g., for Windows environments by specifying a policy in the Domain Controller[2].

The default state of this defense is FALSE as USB AutoRun often is seen as a useful functionality worth keeping.

22. Physical Zone

While physical security [78] (e.g., burglar alarms, fences, locks and doors) naturally are of relevance to cyber security, CySeMoL focuses on cyber-attacks

[2] http://support.microsoft.com/kb/967715

and consequently do not involve *how* physical attacks can be performed in detail. However, it includes modeling *that* they can be performed and *what* attack steps they can enable. This is modeled using the asset `PhysicalZone`, which specifies what assets that an attacker has physical access to during an attack.

`PhysicalZone` can be connected to two different assets (cf. Figure 34). Connection to a `NetworkZone` means that all equipment within the zone is within the same physical environment. Connection to an `OperatingSystem` means that the physical access of this particular system is possible from that particular physical zone. To enable attacks involving a physical zone, there is a need to connect the `Attacker` to this particular zone.

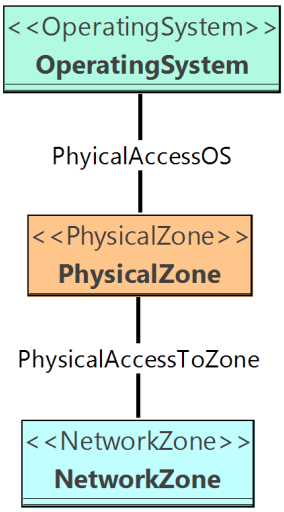

Figure 34: An overview of the connections for `PhysicalZone`.

There is one attack step corresponding to physical zones; this is shown in Table 39 and desribed in in the following section. The reference in Table 39 describe the rationale behind it.

Table 39: Attack steps and defenses of the `PhysicalZone` class.

Attribute	Rationale
Attack step	
`Access`	[78]

22.1. Attack Steps

22.1.1. Access

This attack step concerns whether an attacker has access to the `PhysicalZone` in question. In practice, many aspects naturally affect the likelihood that an attacker can gain physical access [78]. However, as explained previously, this is out of scope for CySeMoL.

The default state of this defense is FALSE (i.e., it is assumed that physical access is not possible for attackers).

23. Access Control Point

An access control point is a place where access can be controlled [79]. This includes evaluating the user's credentials and privileges and granting or denying access. It binds an authentication mechanism to user accounts and to an object (a software). It will maintain access control unless it is bypassed.

An `AccessControlPoint` can be connected to ten different assets (cf. Figure 35). A connection to a `Firewall`, `IDSSensor`, `IPS`, `WebApplicationFirewall`, `WebApplication`, `ApplicationServer`, `ApplicationClient` or `OperatingSystem` denotes a means of logical access for this particular asset. Connection to `PasswordAccount` designates that this particular account can be used to bypass the access control point. Connection to a `PasswordAuthenticationMechanism` designates that accounts related to this access control point are prospect to the security policies of this particular password authentication mechanism.

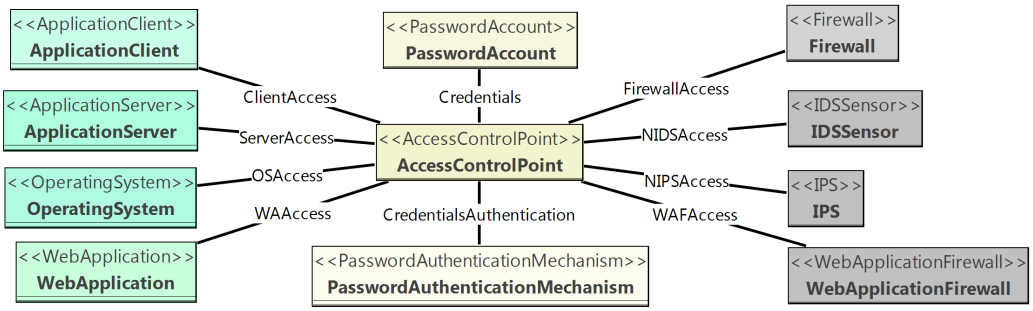

Figure 35: An overview of the connections for `AccessControlPoint`.

There are two attack steps related to access control points; these are shown in Table 40 and detailed in the following section. The references in Table 40 describe the rationale behind them.

Table 40: Attack steps and defenses of the AccessControlPoint class.

Attribute	Rationale
Attack Step	
Bypass	[79]
Interface	[79]

23.1. Attack Steps

23.1.1. Bypass

Bypass concerns whether an attacker is able to bypass the AccessControlPoint and gain access to the system it protects [79].

For this attack step to be successful, there is a need for the attacker to be able to reach it (i.e., Interface = TRUE). If the attacker can reach an access control point, there are four means of bypassing it:

1. If no specific credentials are required.
2. By harvesting the database containing passwords and user accounts corresponding to it and, if required, breaking the existing encryption mechanism (e.g., using John the Ripper[3]).
3. By guessing a correct username and password through some online methodology.
4. By social engineering relevant credentials from an individual.

In CySeMoL, these correspond to:

1. If no PasswordAuthenticationMechanism is connected to the access control point or if PasswordAuthenticationMechanism.Functioning = FALSE .
2. PasswordAccount.GuessAuthenticationCodesOffline.
3. PasswordAccount.GuessAuthenticationCodesOnline.
4. PasswordAccount.SocialEngineerAuthenticationCodes.

If any of these attack steps are TRUE, this attack step is TRUE; else, it is FALSE.

[3] www.openwall.com/john/

23.1.2. Interface

`Interface` concerns whether an attacker is able to reach the `AccessControlPoint` [79]. In CySeMoL, there are two primary means for an attacker to accomplish this attack step: by having physical access to the system it corresponds to, or by being able to connect to the system it corresponds to. The prior includes physical access of a related operating system or network zone; the latter includes being able to connect to the software or device itself, through any means available in CySeMoL. For instance, access of an operating system enables interfacing with access control points of software clients running on the system, and if the attacker can gain an IP on a network zone from where remote access of a firewall is possible, the attacker can reach the access control point of this firewall. An overview of these are described in the list below.

- `PhysicalZone.Access` (of a `NetworkZone` or `OperatingSystem`)
- `ApplicationServer.ConnectTo`
- `OperatingSystem.Access` (to interface with an `ApplicationClient`)
- `NetworkZone.ObtainOwnAddress` (for an `IDSSensor`, `IPS`, `Firewall`, `WAF` or `ApplicationClient` that are connected to the `NetworkZone`)

If an `AccessControlPoint` can be reached by the attacker through any of these means, this attack step is TRUE; else, it is FALSE.

24. Password Authentication Mechanism

A `PasswordAuthenticationMechanism` represents the module that inspects supplied credentials and grant or deny access [80].

A `PasswordAuthenticationMechanism` can be connected to a single type of asset, the `AccessControlPoint` (cf. Figure 36). This connection associates the properties of the authentication mechanism to all `PasswordAccounts` connected to that access control point. The password database corresponding to an authentication mechanism is included within it; there is no need to model it separately.

There are six defenses and one attack step corresponding to password authentication mechanisms; these are shown in Table 41 and desribed in depth in the following sections. The references in Table 41 describe the rationale behind these attributes.

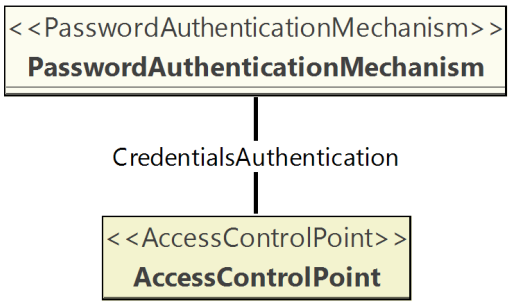

Figure 36: An overview of the connections for `PasswordAuthenticationMechanism`.

Table 41: Attack steps and defenses of the `PasswordAuthenticationMechanism` class.

Attribute	Rationale
Defense	
`Functioning`	[81]
`BackoffTechnique`	[80]
`DefaultPasswordsRemoved`	[81]
`ProactivePasswordChecker`	[82, 81]
`HashedRepository`	[81]
`HashedRepositorySalted`	[83]
Attack step	
`ExtractPasswordRepository`	[84]

24.1. Defenses

24.1.1. Functioning

This defense concerns whether the authentication mechanism is enabled at all, i.e., whether a password is required to bypass the corresponding `AccessControlPoint` or not [81].

The default state of this defense is TRUE.

24.1.2. Backoff Technique

A `BackoffTechnique` increases the difficulty of online password guessing by reacing to consecutive failed login requests. According to [80], there are four types of backoff techniques:

1. *Exponential backoff*: If n is the number of login failures made and x is a predefined number the system waits xn seconds before processing the login

request.

2. *Disconnection*: This technique breaks the connection after x number of failed login attempts. This technique is effective if it takes effort and/or time to reestablish the connection.
3. *Disabling*: With this technique the account is disabled after x number of failed login attempts. The system administrator must then be involved to reactivate the account.
4. *Jailing*: With this technique the system grant access also when login attempts fail. However, access is only granted to some limited part of the system. The threat agents activities attentions can then be examined, or the threat agents time can be wasted.

if any of these techniques are present, `BackoffTechnique` should be defined as TRUE. TRUE is also the default value of this defense.

24.1.3. Default Passwords Removed

Software products frequently come with default passwords that often are easy to obtain (e.g., written in the standard software manual). Best practice is naturally to remove these default passwords and replace them with new, instance specific, passwords [81]. However, this practice is not always applied in practice [85]. This defense concerns whether default passwords are removed or not.

The default state of this defense is TRUE; i.e., it is assumed that default passwords have been removed.

24.1.4. Proactive Password Checker

A proactive password checker studies whether passwords follow some predefined policy [82, 81]. Such a policy could for example be that passwords should be longer than eight characters, contain certain types of symbols, be changed with some frequency or be different from previous passwords. In CySeMoL, it is assumed that any password related to the proactive password checker has at least 8 characters with one special sign, one uppercase, one lowercase letters and one number.

The default state of this defense is TRUE as it is the default in most environments.

24.1.5. Hashed Repository

In a hashed repository no passwords are stored in clear text [81]. Instead, a c hash sum generated using a cryptographic hash function is used to store the

password. Examples of functions include: RIPEMD-128, HAVAL, and SHA-1. CySeMoL does not detail the function used, only the fact that a cryptographic hash function is used.

The default state of this defense is FALSE as many database solutions still store passwords in clear text.

24.1.6. Hashed Repository Salted

To make password cracking more difficult the password is sometimes "salted" before it is hashed [83]. Password cracking often employs dictionaries to identify passwords from their hash sum. A salt has the purpose to make password cracking more difficult by appending some extra characters to the password before passing it to the hash function. By doing so, the hash sum becomes longer and more difficult to guess. This is true for both traditional brute force attacks and for attacks employing rainbow tables.

The default state of this defense is FALSE as salt is not available for Windows operating system user passwords.

24.2. Attack Steps

24.2.1. Extract Password Repository

This attack step involves whether the attacker is able to gain (and process) a local copy of the password repository; a requirement for offline guessing attacks.

To succeed with this attack step there is first a need for the attacker to be able to connect to the application control point related to the authentication mechanism (`AccessControlPoint.Interface` = TRUE). If so, the probability of successful attack is 5%. This number was estimated by a security expert and refers to the likelihood that there is a vulnerability in the authentication function that enables password extraction [84]. For instance, CVE-2008-4037 is such a vulnerability and concerns Microsoft's SMB implementation. Else, this attack step is FALSE.

25. Password Account

A `PasswordAccount` is a user account protected by a password; the far most common authentication mechanism in information technology.

Two assets can be connected to a `PasswordAccount` (cf. Figure 37). Connection to a `Person` specifies that this individual has access to that particular account. Connection to a `PasswordAuthenticationMechanism` denotes that the account is protected by that particular authentication mechanism.

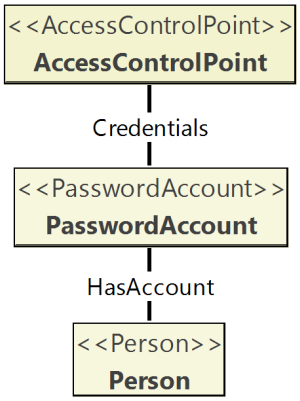

Figure 37: An overview of the connections for `PasswordAccount`.

There are three attack steps corresponding to password accounts; these are shown in Table 42 and desribed in depth in the following sections. The references in Table 42 describe the rationale behind these attributes.

Table 42: Attack steps and defenses of the `PasswordAccount` class.

Attribute	**Rationale**
Attack step	
`GuessAuthenticationCodesOffline`	[84]
`GuessAuthenticationCodesOnline`	[84, 15]
`SocialEngineerAuthenticationCodes`	[86, 31, 87, 88]

25.1. Attack Steps

25.1.1. Guess Authentication Codes Offline

`GuessAuthenticationCodesOffline` concerns password cracking; retrieving credentials from a locally accessible password repository [83].

For many authentication mechanisms this procedure can be completely automated under certain conditions. For example, Ophcrack is a Windows XP based "password recovery tool" which finds 99.9% of all alphanumeric passwords shorter than 15 characters in a matter of seconds[4]. Ophcrack, and similar tools,

[4]http://ophcrack.sourceforge.net/

however require access to the password file. In CySeMoL, this corresponds to `PasswordAuthenticationMechanism.ExtractPasswordRepository`, which needs to have the state TRUE for this attack to be able to succeed.

If this is the case, the probability that an account password can be cracked depends on its entropy. Tools like Ophcrack are typically limited with respect to password length, characters in passwords, or some combination of these. Hence, long passwords with many types of characters (number, alphabetical, special characters) will make the account significantly more difficult to crack. In CySeMoL, three defenses concern these aspects:

1. `PasswordAuthenticationMechanism.ProactivePasswordChecker` (PC)
2. `PasswordAuthenticationMechanism.HashedRepository` (HR)
3. `PasswordAuthenticationMechanism.HashedRepositorySalted` (HRS)

The effectiveness of these defenses were estimated using data on the speed of cracking tools for different types of passwords and entropy, along with assumptions regarding presumed characteristics of passwords and entropy given these defenses [84]. In short, this findings of this study show that a `ProactivePasswordChecker` in practice next to completely mitigates the attack (i.e., if it is TRUE, this attack step is FALSE). Furthermore, a scenario with a salted hashed repository (i.e., HashedRepository = TRUE and HashedRepositorySalted = TRUE) also mitigates the attack. Given use of hash without salt, the likelihood of success is approximately 60%. In other cases, the success rate of this attack step is TRUE. This is shown in Table 43.

25.1.2. Guess Authentication Codes Online

This attack step include attacks that concern correctly guessing credentials using a live application, i.e., "online-attacks" [89].

The difficulty of guessing a password online depends on the search space an attacker is confronted with [180]. For example, authentication mechanisms comprising of four digits are easier to guess than those that that comprise of an arbitrary number of alphanumerical characters. For specific types of authentication mechanisms guessing can be supported by traces left by legitimate users, e.g. frequently used buttons on a key panel. In CySeMoL, a password however concerns a password in the "traditional" sense, i.e., a string of arbitrary length that can constist of a mix of upper case letters, lower case letters, numbers and special characters.

Table 43: Defenses affecting likelihood of `GuessAuthenticationCodesOffline`.

PC	HR	HRS	Data
TRUE	TRUE	TRUE	bernoulli(0)
TRUE	TRUE	FALSE	bernoulli(0)
TRUE	FALSE	TRUE	bernoulli(0)
TRUE	FALSE	FALSE	bernoulli(0)
FALSE	TRUE	TRUE	bernoulli(0)
FALSE	TRUE	FALSE	bernoulli(0.6)
FALSE	FALSE	TRUE	bernoulli(1)
FALSE	FALSE	FALSE	bernoulli(1)

In CySeMoL, online password guessing first of all requires that the attacker can reach the authentication mechanism (i.e., `AccessControlPoint.Interface` = TRUE). If so, the likelihood of successful attack depends on the presence of four defenses:

1. If default passwords have been removed.
2. If a proactive password checker is employed.
3. If a back-off technique is employed.
4. If an authenticated or unauthenticated scan has been perfomed by a network vulnerability scanner

In CySeMoL, these concern the following defenses:

1. `PasswordAuthenticationMechanism.DefaultPasswordsRemoved` (DPR)
2. `PasswordAuthenticationMechanism.ProactivePasswordChecker` (PC)
3. `PasswordAuthenticationMechanism.BackoffTechnique` (BOT)
4. `NetworkVulnerabilityScanner.Functioning` (NVS)

The effectiveness of these defenses were estimated through a literature study [84] and an experiment [15], using the same assumptions regarding the type of passwords involved as for `GuessAuthenticationCodesOffline`. A vulnerability scanner tests passwords that have been manually input by the user; thus, in practice they serve to mitigate the possibility of default passwords. The likelihood of success of this attack given the presence or absence of these defenses is illustrated by Table 44. See [84] for detailed information on how these estimates were derived.

Table 44: Defenses affecting likelihood of `GuessAuthenticationCodesOnline`.

PC	DPR or NVS	BOT	Data
TRUE	TRUE	TRUE	bernoulli(0)
TRUE	TRUE	FALSE	bernoulli(0)
TRUE	FALSE	TRUE	bernoulli(1)
TRUE	FALSE	FALSE	bernoulli(1)
FALSE	TRUE	TRUE	bernoulli(0.0001)
FALSE	TRUE	FALSE	bernoulli(0.6)
FALSE	FALSE	TRUE	bernoulli(1)
FALSE	FALSE	FALSE	bernoulli(1)

25.1.3. Social Engineer Authentication Codes

A social engineering attack involves deceiving an individual into complying with a malicious request [90]. A social engineering attack can be carried out through any means, with an especially common variant being by email (often called phishing). This attack step concerns an attacker that is able to social engineer credentials of a targeted application through *any* means.

To be successful with this attack step, the attacker must be able to connect to the targeted application (`ApplicationControlPoint.Interface` = TRUE). Furthermore, the desired credentials needs to have some active user that can be deceived (i.e., `PasswordAccount` needs to be connected to a `Person`). If this is the case, the likelihood of success depends on whether the targeted user has undergone security awareness training. The likelihood of success given different states of security awareness training are given below:

- `SecurityAwarenessProgram.Functioning` = TRUE : bernoulli(exp(0.0715,Attacker.Time))
- `SecurityAwarenessProgram.Functioning` = FALSE : bernoulli(exp(0.241,Attacker.Time))

These estimates are derived from [86, 31, 87, 88].

26. Person

A `Person` is an individual, or type of individual, who use IT in some means.

A Person can be connected to three assets (cf. Figure 38). A connection to a PasswordAccount denotes that this individual has access to the credentials of that particular account. Connection to a SocialZone denotes that this individual is part of that particular group of individuals who socialize in some means. Connection to a SecurityAwarenessProgram denotes that the individual is recipient to that particular security training program.

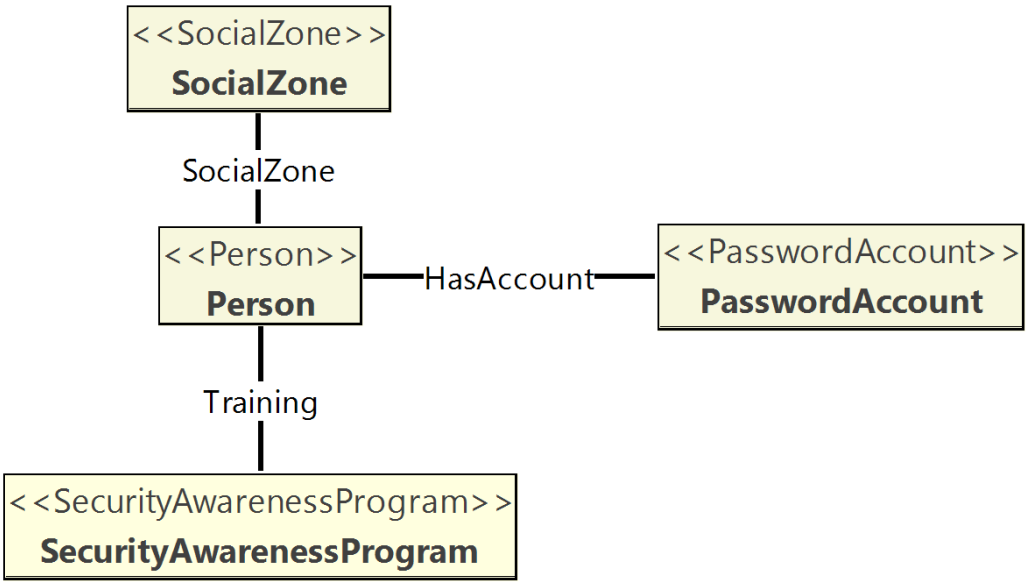

Figure 38: An overview of the connections for Person.

There are no defenses or attacks corresponding to individuals in CySeMoL (these are instead modeled through PasswordAccount.SocialEngineerAuthenticationCodes or OperatingSystem.AccessThroughPortableMedia using SocialZone).

27. Social Zone

SocialZone denotes a group of individuals who are prone to sharing documents and devices, e.g., a work-group in an office space. SocialZone enables modeling attacks against IT-wise isolated devices (which can is the case in information-critical environments, e.g., critical infrastructure control systems). In practice, this is managed by relating Access of an OperatingSystem to AccessThroughPortableMedia of other OperatingSystems that have local users (Persons) who share the same SocialZone.

SocialZone can be connected to a single type of asset - any Person that is part of it (cf. Figure 39).

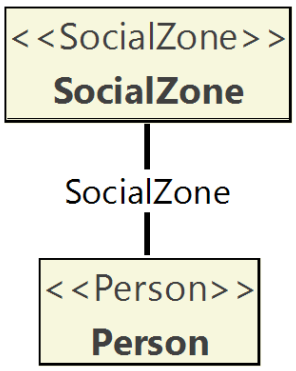

Figure 39: An overview of the connections for SocialZone.

There is one attack step corresponding to social zones in CySeMoL (cf. Table 45). This attack step is described in the following section.

Table 45: Attack steps and defenses of the SocialZone class.

Attribute	Rationale
Attack Step	
SharePortableMedia	[91]

27.1. Attack Steps

27.1.1. Share Portable Media

This attack step concerns the possibility that individuals within the same social zone share portable media (e.g., a USB drive), a very common type of attack vector [91].

This attack step is TRUE if a Person connected to the SocialZone has a PasswordAccount related to an OperatingSystem that has been compromised by the attacker (i.e., if OperatingSystem.Access = TRUE). It is FALSE in other cases. The likelihood of this overall attack vector being TRUE is expressed through OperatingSystem.AccessThroughPortableMedia.

28. Security Awareness Program

Security awareness and training programs are crucial for enabling users with the knowledge required to react to security threats [92]. NIST states that the activity is "the vehicle for disseminating information that users, including managers, need in order to do their jobs" [93]. In CySeMoL, this process is encounted for by the class `SecurityAwarenessProgram`.

A `SecurityAwarenessProgram` has a single type of connection - the `Persons` that it concerns (cf. Figure 40).

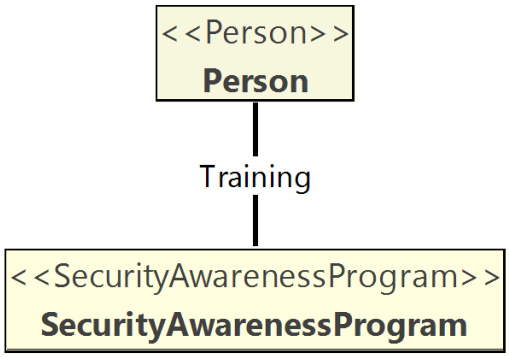

Figure 40: An overview of the connections for `SecurityAwarenessProgram`.

There is one defenses corresponding to security awareness programs; this is shown in Table 46 and described in the following section. The references in Table 46 describe the rationale behind it.

Table 46: Attack steps and defenses of the `SecurityAwarenessProgram` class.

Attribute	Rationale
Defense	
`Functioning`	[93]

28.1. Defenses

28.1.1. Functioning

This attributes concerns whether the awareness program reaches out to the persons that are included in it. In other words, if the intended users participate

in its designated activities. For instance, whether or not they participate in its seminars or read its required material [93].

The default state of this defense is TRUE (i.e., it is assumed that security training is conducted; else, it should not be modeled).

29. Example model

To view screen casts for how to use CySeMoL, please go `www.ics.kth.se/cysemol`.

References

[1] A. J. A. Wang, Information security models and metrics, in: Proceedings of the 43rd annual Southeast regional conference-Volume 2, ACM, 2005, pp. 178–184.

[2] CCRA, Common Criteria for Information Technology Security Evaluation, Available on `http://www.commoncriteriaportal.org/`, accessed June 24, 2013 (2012).

[3] C. Alberts, A. Dorofee, J. Stevens, C. Woody, Introduction to the octave approach, Pittsburgh, PA, Carnegie Mellon University.

[4] F. den Braber, I. Hogganvik, M. S. Lund, K. Stølen, F. Vraalsen, Model-based security analysis in seven steps-a guided tour to the coras method, BT Technology Journal 25 (1) (2007) 101–117.

[5] R. Breu, F. Innerhofer-Oberperfler, A. Yautsiukhin, Quantitative assessment of enterprise security system, in: Availability, Reliability and Security, 2008. ARES 08. Third International Conference on, IEEE, 2008, pp. 921–928.

[6] H. Huang, S. Zhang, X. Ou, A. Prakash, K. Sakallah, Distilling critical attack graph surface iteratively through minimum-cost sat solving, in: Proceedings of the 27th Annual Computer Security Applications Conference, ACSAC '11, ACM, New York, NY, USA, 2011, pp. 31–40. `doi:10.1145/2076732.2076738`.
URL `http://doi.acm.org/10.1145/2076732.2076738`

[7] X. Ou, W. F. Boyer, M. A. McQueen, A scalable approach to attack graph generation, in: Proceedings of the 13th ACM conference on Computer and communications security, CCS '06, ACM, New York, NY, USA, 2006, pp. 336–345. doi:10.1145/1180405.1180446.
URL http://doi.acm.org/10.1145/1180405.1180446

[8] K. Ingols, M. Chu, R. Lippmann, S. Webster, S. Boyer, Modeling modern network attacks and countermeasures using attack graphs, in: Computer Security Applications Conference, 2009. ACSAC '09. Annual, 2009, pp. 117–126. doi:10.1109/ACSAC.2009.21.

[9] S. Jajodia, S. Noel, B. O'Berry, Topological analysis of network attack vulnerability, in: Managing Cyber Threats, Springer, 2005, pp. 247–266.

[10] T. Sommestad, M. Ekstedt, H. Holm, The cyber security modeling language: A tool for assessing the vulnerability of enterprise system architectures, Systems Journal, IEEE PP (99) (2012) 1. doi:10.1109/JSYST.2012.2221853.

[11] P. Johnson, J. Ullberg, M. Buschle, U. Franke, K. Shahzad, P2amf: Predictive, probabilistic architecture modeling framework, in: International IFIP Working Conference on Enterprise Interoperability Information, Services and Processes for the Interoperable Economy and Society, 2013.

[12] O. Uml, 2.0 OCL Specification, OMG Adopted Specification (ptc/03-10-14).

[13] S. Liu, B. Cheng, Cyberattacks: Why, what, who, and how, IT professional 11 (3) (2009) 14–21.

[14] T. Gerace, H. Cavusoglu, The critical elements of the patch management process, Communications of the ACM 52 (8) (2009) 117–121.

[15] H. Holm, Performance of automated network vulnerability scanning at remediating security issues, Computers & Security 31 (2) (2012) 164–175.

[16] S. Whalen, An introduction to arp spoofing, Node99 [Online Document], April.

[17] R. Oppliger, Internet security: firewalls and beyond, Communications of the ACM 40 (5) (1997) 92–102.

[18] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, D. Boneh, On the effectiveness of address-space randomization, in: Proceedings of the 11th ACM conference on Computer and communications security, ACM, 2004, pp. 298–307.

[19] C. Cowan, F. Wagle, C. Pu, S. Beattie, J. Walpole, Buffer overflows: Attacks and defenses for the vulnerability of the decade, in: DARPA Information Survivability Conference and Exposition, 2000. DISCEX'00. Proceedings, Vol. 2, IEEE, 2000, pp. 119–129.

[20] J. Hruska, Computer viruses and anti-virus warfare, Ellis Horwood, 1992.

[21] M. Al-Zarouni, The reality of risks from consented use of usb devices.

[22] T. Grance, J. Hash, S. Peck, J. Smith, K. Korow-Diks, Security guide for interconnecting information technology systems, NIST Special Publication 800 (47).

[23] V. D. Gligor, A note on denial-of-service in operating systems, Software Engineering, IEEE Transactions on (3) (1984) 320–324.

[24] T. Sommestad, H. Holm, M. Ekstedt, Estimates of success rates of denial-of-service attacks, in: Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, IEEE, 2011, pp. 21–28.

[25] T. Sommestad, M. Ekstedt, H. Holm, M. Afzal, Security mistakes in information system deployment projects, Information Management & Computer Security 19 (2) (2011) 80–94.

[26] A. Wool, A quantitative study of firewall configuration errors, Computer 37 (6) (2004) 62–67.

[27] T. Sommestad, Exploiting network configuration mistakes: practitioners self-assessed success rate, Royal Instit. Technol., Tech. Rep. TRITA-EE 69.

[28] M. A. McQueen, W. F. Boyer, M. A. Flynn, G. A. Beitel, Time-to-compromise model for cyber risk reduction estimation, in: First Workshop on Quality of Protection, 2005.

[29] T. Sommestad, H. Holm, M. Ekstedt, Estimates of success rates of remote arbitrary code execution attacks, Information Management & Computer Security 20 (2) (2012) 107–122.

[30] H. Holm, T. Sommestad, U. Franke, M. Ekstedt, Success rate of remote code execution attacks–expert assessments and observations, Journal of Universal Computer Science 18 (6) (2012) 732–749.

[31] J. R. Jacobs, Measuring the effectiveness of the usb flash drive as a vector for social engineering attacks on commercial and residential computer systems, Ph.D. thesis, Embry Riddle Aeronautical University (2011).

[32] H. H. E. M. Sommestad, Teodor, N. Honeth, Quantifying the effectiveness of intrusion detection systems in operation through domain experts.

[33] H. Holm, Signature based intrusion detection for zero-day attacks: (not) a closed chapter?, in: 47th Hawaii International Conference on System Science (HICSS), Submitted.

[34] T. Sommestad, A. Hunstad, Intrusion detection and the role of the system administrator, Information Management & Computer Security 21 (1) (2013) 30–40.

[35] J. A. Morales, R. Sandhu, S. Xu, Evaluating detection and treatment effectiveness of commercial anti-malware programs, in: Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on, IEEE, 2010, pp. 31–38.

[36] Baggett, Mark, Effectiveness of Antivirus in Detecting Metasploit Payloads , Available on `http://www.sans.org/reading_room/whitepapers/casestudies/effectiveness-antivirus-detecting-metasploit-payloads_2134`, accessed April 19, 2013 (2008).

[37] G. McGraw, Software security, Security & Privacy, IEEE 2 (2) (2004) 80–83.

[38] V. Thomas, P. Ramagopal, R. Mohandas, The rise of autorun-based malware, McAfee Avert Labs., McAfee Inc.

[39] D. V. Pham, M. N. Halgamuge, A. Syed, P. Mendis, Optimizing windows security features to block malware and hack tools on usb storage devices, in: Progress in electromagnetics research symposium, 2010.

[40] J. Wilander, M. Kamkar, A comparison of publicly available tools for dynamic buffer overflow prevention., in: NDSS, Vol. 3, 2003, pp. 149–162.

[41] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, C. Kruegel, A view on current malware behaviors, in: USENIX workshop on large-scale exploits and emergent threats (LEET), 2009.

[42] G. Schryen, Security of open source and closed source software: An empirical comparison of published vulnerabilities.

[43] J. Novak, A. Krajnc, R. Zontar, Taxonomy of static code analysis tools, in: MIPRO, 2010 Proceedings of the 33rd International Convention, IEEE, 2010, pp. 418–422.

[44] A. Ozment, Improving vulnerability discovery models, in: Proceedings of the 2007 ACM workshop on Quality of protection, ACM, 2007, pp. 6–11.

[45] H. Holm, M. Korman, M. Ekstedt, A markovian model for likelihood estimations of acquirement of critical software vulnerabilities and exploits.

[46] H. Holm, M. Buschle, R. Lagerström, M. Ekstedt, Automatic data collection for enterprise architecture models, Software & Systems Modeling (2012) 1–17.

[47] T. Sommestad, H. Holm, M. Ekstedt, Effort estimates for vulnerability discovery projects, in: System Science (HICSS), 2012 45th Hawaii International Conference on, IEEE, 2012, pp. 5564–5573.

[48] C. Cowan, Software security for open-source systems, Security Privacy, IEEE 1 (1) (2003) 38 – 45. doi:10.1109/MSECP.2003.1176994.

[49] H. Holm, M. Ekstedt, Estimates on the effectiveness of web application firewalls for targeted attacks, Information Management and Computer Security (2013) 5029 – 5038.

[50] D. Mitropoulos, V. Karakoidas, P. Louridas, D. Spinellis, Countering code injection attacks: a unified approach, Information Management & Computer Security 19 (3) (2011) 177–194.

[51] H. Holm, M. Ekstedt, A metamodel for web application injection attacks and countermeasures, in: Trends in Enterprise Architecture Research and Practice-Driven Research on Enterprise Transformation, Springer, 2012, pp. 198–217.

[52] R. L. Jones, A. Rastogi, Secure coding: building security into the software development life cycle, Information Systems Security 13 (5) (2004) 29–39.

[53] Y. Shin, L. A. Williams, Towards a taxonomy of techniques to detect cross-site scripting and sql injection vulnerabilities.

[54] J. Fonseca, M. Vieira, H. Madeira, The web attacker perspective-a field study, in: Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on, IEEE, 2010, pp. 299–308.

[55] H. Holm, M. Ekstedt, T. Sommestad, Effort estimates on web application vulnerability discovery, in: 46th Hawaii International Conference on System Science (HICSS), IEEE, 2013, pp. 5029 – 5038.

[56] K. Scarfone, P. Mell, Guide to intrusion detection and prevention systems (idps), NIST Special Publication 800 (2007) (2007) 94.

[57] E. Shmueli, R. Vaisenberg, Y. Elovici, C. Glezer, Database encryption: an overview of contemporary challenges and design considerations, ACM SIGMOD Record 38 (3) (2010) 29–34.

[58] R. Ramakrishnan, J. Gehrke, Database management systems, Osborne/McGraw-Hill, 2000.

[59] D. Litchfield, C. Anley, J. Heasman, B. Grindlay, The database hacker's handbook, Wiley, 2005.

[60] D. E. Bell, L. J. La Padula, Secure computer system: Unified exposition and multics interpretation, Tech. rep., DTIC Document (1976).

[61] P. Syverson, A taxonomy of replay attacks [cryptographic protocols], in: Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings, IEEE, 1994, pp. 187–191.

[62] B. Harris, R. Hunt, Tcp/ip security threats and attack methods, Computer Communications 22 (10) (1999) 885–897.

[63] Y. Desmedt, Man-in-the-middle attack, in: Encyclopedia of Cryptography and Security, Springer, 2005, pp. 368–368.

[64] J. A. Clark, J. L. Jacob, Protocols are programs too: the meta-heuristic search for security protocols, Information and Software Technology 43 (14) (2001) 891–904.

[65] B. C. Neuman, S. G. Stubblebine, A note on the use of timestamps as nonces, ACM SIGOPS Operating Systems Review 27 (2) (1993) 10–14.

[66] A. Chakrabarti, G. Manimaran, Internet infrastructure security: A taxonomy, Network, IEEE 16 (6) (2002) 13–21.

[67] V. Goyal, R. Tripathy, An efficient solution to the arp cache poisoning problem, in: Information Security and Privacy, Springer, 2005, pp. 40–51.

[68] R. Von Solms, Information security management: why standards are important, Information Management & Computer Security 7 (1) (1999) 50–58.

[69] M. S. Desai, T. C. Richards, T. von der Embse, System insecurity–firewalls, Information management & computer security 10 (3) (2002) 135–139.

[70] K. Scarfone, P. Hoffman, Guidelines on firewalls and firewall policy, NIST Special Publication 800 (2009) 41.

[71] E. Biermann, E. Cloete, L. M. Venter, A comparison of intrusion detection systems, Computers & Security 20 (8) (2001) 676–683.

[72] M. A. Faysel, S. S. Haque, Towards cyber defense: Research in intrusion detection and intrusion prevention systems, IJCSNS International Journal of Computer Science and Network Security 10 (7) (2010) 316–325.

[73] S. Welberg, Vulnerability management tools for cots software-a comparison.

[74] NERC, Nerc cip 002-009, Tech. rep., NERC (2007).

[75] K. Stouffer, J. Falco, K. Scarfone, Guide to industrial control systems (ics) security, NIST Special Publication 800 (82) (2008) 16–16.

[76] J. Wack, M. Tracy, M. Souppaya, Guideline on network security testing, Nist special publication 800 (2003) 42.

[77] D. Longley, et al., Information security management and modelling, Information Management & Computer Security 7 (1) (1999) 30–40.

[78] L. Fennelly, Effective physical security, Butterworth-Heinemann, 2012.

[79] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, Role-based access control models, Computer 29 (2) (1996) 38–47.

[80] M. Bishop, Computer security: Art and science. 2003, Westford, MA: Addison Wesley Professional (2003) 4–12.

[81] K. Scarfone, M. Souppaya, Guide to enterprise password management, NIST Special Publication 800 (2009) 118.

[82] J. J. Yan, A note on proactive password checking, in: Proceedings of the 2001 workshop on New security paradigms, ACM, 2001, pp. 127–135.

[83] S. Marechal, Advances in password cracking, Journal in computer virology 4 (1) (2008) 73–81.

[84] Sommestad, Teodor, Password authentication attacks: a survey of attacks and when they will succeed, institution = Royal Institute of Technology, year = 2011, type = Tecnhical report, number = TRITA-EE 2011:067, July,, Tech. rep.

[85] W. C. Summers, E. Bosworth, Password policy: the good, the bad, and the ugly, in: Proceedings of the winter international synposium on Information and communication technologies, Trinity College Dublin, 2004, pp. 1–6.

[86] T. N. Jagatic, N. A. Johnson, M. Jakobsson, F. Menczer, Social phishing, Communications of the ACM 50 (10) (2007) 94–100.

[87] S. Stasiukonis, Social engineering, the usb way, Dark Reading 7.

[88] R. Dodge, A. Ferguson, Using phishing for user email security awareness, Security and Privacy in Dynamic Environments (2006) 454–459.

[89] W. E. Burr, D. F. Dodson, W. T. Polk, Electronic authentication guideline, US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2004.

[90] M. Workman, Wisecrackers: A theory-grounded investigation of phishing and pretext social engineering threats to information security, Journal of the American Society for Information Science and Technology 59 (4) (2008) 662–674.

[91] U. When New, Understanding usb malware.

[92] E. Schultz, Security training and awarenessfitting a square peg in a round hole, Computers & Security 23 (1) (2004) 1–2.

[93] M. Wilson, J. Hash, Building an information technology security awareness and training program, NIST Special publication 800 (2003) 50.